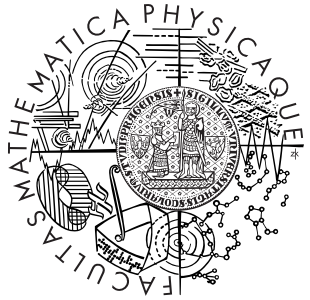
UNIVERZITA KARLOVA V PRAZE
MATEMATICKO-FYZIKÁLNÍ FAKULTA

DIPLOMOVÁ PRÁCE



Štefan Čudai

Extrakce objektů z komplexních obrazových scén

Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. RNDr. Tomáš Skopal, Ph.D.
Studijní obor: Softwarové systémy

2010

Rád by som poďakoval doc. RNDr. Tomášovi Skopalovi, Ph.D. za vedenie diplomovej práce, za ochotu, trpezlivosť a čas strávený pri konzultácii. Ďalej by som chcel poďakovať Mgr. Marte Lungovej za konzultácie, trpezlivosť a jazykovú korektúru. V neposlednej rade chcem poďakovať svojim rodičom za podporu počas celého štúdia.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 6.8.2010 Štefan Čudai

Obsah

1 Úvod **6**
- 1.1 Cieľ práce . . . 7
- 1.2 Prehľad kapitol . . . 7

2 Databázy digitálnych obrázkov **9**
- 2.1 Vyhľadávanie obrázkov . . . 9
- 2.2 CBIR systémy . . . 10
- 2.3 Detektor a deskriptor výrazných častí obrazu . . . 12
- 2.4 Metódy detekcie hrán . . . 13
  - 2.4.1 Sobelov detektor hrán . . . 14

3 Analýza a návrh **17**
- 3.1 Prehľad . . . 17
- 3.2 Deskriptor . . . 17
  - 3.2.1 Dekompozícia obrázku na bunky . . . 18
  - 3.2.2 Deskriptor hranového histogramu . . . 19
- 3.3 Algoritmus vyhľadávania . . . 21
- 3.4 Vylepšený algoritmus vyhľadávania . . . 22
  - 3.4.1 Gradienty . . . 22
  - 3.4.2 Prázdne bunky . . . 23
  - 3.4.3 Posúvanie mriežky . . . 23
  - 3.4.4 Veľmi podobné bunky . . . 23
  - 3.4.5 Zašumené bunky a obrázky . . . 25

4 Implementácia **27**
- 4.1 Technológie . . . 27
- 4.2 Dataset . . . 27
  - 4.2.1 Vytváranie datasetu . . . 28
  - 4.2.2 Načítanie datasetu . . . 28
  - 4.2.3 Štruktúra súborov . . . 28
- 4.3 Implementácia SBIR systému . . . 30
  - 4.3.1 Objektový model deskriptoru . . . 30
  - 4.3.2 Objektový model datasetu . . . 32
- 4.4 Operácie . . . 33
  - 4.4.1 Operácia vytvorenia datasetu . . . 33
  - 4.4.2 Operácia načítania datasetu . . . 35

4.4.3 Operácia vyhľadávania 35
4.5 Kresliace plátno 36
4.6 Nástroj pre experimentovanie 36

5 Užívateľská dokumentácia **38**
5.1 Dotazovací a experimentálny režim 38
5.2 Hlavné okno aplikácie 39
5.3 Vytvorenie a načítanie datasetu 44
5.4 Dialógové okno pre experimentálny režim 44

6 Experimenty **50**
6.1 Testovacie prostredie 50
6.2 Metódy testovania 51
6.3 Výsledky testovania 51
6.3.1 Konvolučné matice 57
6.3.2 Rozpoznávanie neštruktúrovaných objektov 58

7 Záver **61**
7.1 Ďalší vývoj 62

Odkazy na DVD **63**

Literatúra **64**

Název práce: Extrakce objektů z komplexních obrazových scén
Autor: Štefan Čudai
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: doc. RNDr. Tomáš Skopal, Ph.D.
e-mail vedoucího: tomas.skopal@mff.cuni.cz

Abstrakt: Cieľom práce bolo navrhnúť a implementovať architektúru pre získavanie obrázkov z databázy na základe ručne nakresleného zadania. Vyhľadávanie je založené na deskriptore, ktorý popisuje čierno-biely obrázok nakreslený užívateľom a farebné obrázky z databázy. Ide o upravenú verziu deskriptoru navrhnutého pre MPEG-7, ktorý je v anglickej literatúre známy pod pojmom *Edge histogram descriptor*, a jeho základom je popis lokálnej distribúcie hrán. Deskriptor a algoritmus navrhnutý v článku [6] bol reimplementovaný a po analýze ich vlastností boli navrhnuté vylepšenia, ktoré boli následne overené v reálnej implementácii. Keďže výsledok vyhľadávania závisí na kvalite vstupného náčrtku, boli navrhnuté spôsoby, ako túto závislosť znížiť - *posúvanie mriežky*, identifikácia *prázdnych buniek*, ktoré radikálne menia výsledok dotazu a detekcia *veľmi podobných buniek*. Na základe identifikácie *zašumených buniek* bola navrhutá metóda detekcie obrázkov so stromami, kríkmi alebo inými prírodnými objektami. V experimentálnej časti sa skúma vplyv rôznych parametrov vyhľadávania, ako sú rozmery mriežky alebo použité konvolučné matice, na presnosť nájdených výsledkov. Experimentálne výsledky demonštrujú, že systém poskytuje užívateľovi intuitívny prístup k databáze obrázkov.

Klíčová slova: CBIR systém, vyhľadávanie obrázkov, vyhľadávanie náčrtkom

Title: Object extraction from complex images
Author: Štefan Čudai
Department: Department of Software Engineering
Supervisor: doc. RNDr. Tomáš Skopal, Ph.D.
Supervisor's e-mail address: tomas.skopal@mff.cuni.cz

Abstract: The aim of this work was to design and implement an architecture for image retrieval based on hand drawn sketches. A descriptor is used to classify a monochrome user-drawn sketch as well as color images in the database. It is based on a descriptor proposed for MPEG-7, a so-called Edge histogram descriptor. Using [6] as reference, the algorithm was implemented, evaluated and improvements were proposed. Because sketch quality is an important factor in the overall search quality, several ways to decrease this dependance were proposed. These are search grid shifting, empty cell detection and very similar cell detection. A method for finding images containing trees and other natural objects using noisy cells was proposed. Experimental part of this work deals with determining the influence of various parameters on the precision of search results. The results obtained demostrate that the system provides an intuitive way to search an image database.

Keywords: CBIR system, image retrieval, sketch-based image retrieval

Kapitola 1

Úvod

Za posledné dve desaťročia sa stúpajúcim tempom zväčšuje objem digitálnych fotografií. Digitálne fotografie majú širokú škálu uplatnenia jak v médiách, reklame, umení ale aj v oblasti vzdelávania. Sila digitálnej fotografie spočíva v jednoduchosti, akou dokáže uchovať veľké množstvo informácií. V závislosti od komplexnosti obrazovej scény je možné z obrázku získať informácie, ktoré by človek popísal až niekoľkými vetami.

Vo veľkom množstve multimediálneho materiálu sa dá len ťažko zorientovať. Ručné prechádzanie rozsiahlych databáz obrázkov alebo prezeranie videí za účelom nájsť to, čo v danej chvíli potrebujeme, je časovo náročné. A preto je žiaduce vytvoriť systémy, ktoré by podobne, ako človek, dokázali rozpoznať, čo sa na obrázku nachádza.

Odborníci z oboru počítačového videnia sa už niekoľko rokov venujú problematike rozpoznávania obrazu, spracovaniu digitálneho obrazu, vyhľadávaniu obrázkov alebo videí v multimediálnych databázach, ale aj iným problémom súvisiacich s digitálnym obrazom. Za túto dobu vzniklo ohromné množstvo algoritmov, ktoré sa snažia obrazovú informáciu spracovať a získať tak potrebnú sémantickú informáciu, ale problém prekonať sémantickú medzeru je náročný.

Súčasné systémy pre vyhľadávanie v obrázkových databázach sú založené na rôznych technikách. Jedna z najviac skúmaných techník predpokladá, že vstupný dotaz od užívateľa je obrázok alebo časť obrázku. Iné zase umožňujú užívateľovi hrubo načrtnúť farebné oblasti, prípadne zadať pomer farieb na obrázku. Tie najjednoduchšie predpokladajú, že každý obrázok v databáze obsahuje textovú anotáciu, na základe ktorej užívateľ zadá svoj dotaz formou kľúčových slov [6].

Predstavme si klasický prípad, v ktorom užívateľ chce zo svojej databázy fotiek alebo kreslených obrázkov nájsť také, ktoré obsahujú v ľavej časti budovu. Na to, aby ich databáza vyhľadala, musí dostať od užívateľa vstup, podľa ktorého nájde najpodobnejšie obrázky. V prípade prvej spomenutej techniky by bol vstupom iný podobný obrázok, ktorý má tiež na ľavej strane budovu alebo iba časť obrázku s budovou. Problém spočíva práve v nutnosti nájsť vhodný obrázok pre vstup. Iný prístup by bol vyhľadať dané obrázky podľa textovej anotácie, ktorá by hovorila o obsahu každého obrázku. Za výraznú nevýhodu tejto metódy je možné považovať nutnosť priradenia anotácie každému obrázku

čo by bolo pri takom množstve existujúcich dát nemysliteľné. Taktiež ľudia z rôznych kultúr môžu obrázok popísať rôzne pre rozdiely v slovnej zásobe. Teda vyhľadávanie podľa textovej anotácie by v konečnom dôsledku nebolo efektívne alebo spoľahlivé.

V článku [6] sa na problém autori pozreli z druhej strany a navrhli deskriptor a algoritmus na vyhľadávanie, ktorý na základe rukou nakresleného náčrtku dokáže nájsť podobné obrázky. Tento prístup vyzerá byť veľmi sľubný vzhľadom na skutočnosť, že pre človeka je omnoho jednoduchšie nakresliť významné čiary, tvary alebo ľubovoľnú časť scény. Táto metóda vyzerá byť pre praktické použitie vhodná, pretože ľudský mozog sa snaží zapamätať si práve tieto významné hrany na obrázku.

1.1 Cieľ práce

Prvým cieľov práce je pochopiť a reimplementovať algoritmus popísaný v článku [6], ktorý bol publikovaný na konferencii *EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009)*. V tomto algoritme je dostatok priestoru pre vylepšenia. Skúmaním vlastností buniek obrázku budú zavedené pojmy ako *zašumená bunka* a *veľmi podobné bunky*. Za ďalšie vylepšenie sú považované *posúvacie mriežky*.

Okrem snahy vylepšiť algoritmus je potrebné identifikovať najlepšie parametre mriežky a určiť vhodnú konvolučnú maticu pre výpočet gradientov. To je cieľom experimentov v závere práce.

Hlavným prínosom práce je implementácia systému založeného na dotazovaní náčrtkom[1].

1.2 Prehľad kapitol

V druhej kapitole bude čitateľ uvedený do problematiky vyhľadávania obrázkov a problémov s tým súvisiacimi. Popísané budú CBIR systémy, detektor, deskriptor a ich vzájomný vzťah. V závere sa sústredíme na jednu konkrétnu metódu detekcie hrán, tzv. Sobelov detektor, ktorý načrtne problematiku výpočtov gradientov.

Tretia kapitola je venovaná analýze problému a návrhu jeho riešenia. Bude definovaný deskriptor popisujúci lokálnu distribúciu hrán. Bude opísaný algoritmus pre vyhľadávanie, vytvorený nad zadefinovaným deskriptorom, ktorý bol navrhnutý v článku [6]. Na základe toho budú demonštrované vylepšenia tohto algoritmu. Zavedú sa pojmy *posúvanie mriežky, veľmi podobné bunky* a *zašumené bunky*.

Štvrtá kapitola sa sústredí na implementačnú časť práce. Popisuje technológie použité na vytvorenie aplikácie. Vysvetľuje význam len tých najvýznamnejších a zaujímavých častí kódu. V kapitole je zavedený pojem *dataset*, ktorý je akousi abstrakciou databázy obrázkov.

Užívateľskej dokumentácii je venovaná piata kapitola. Na jej začiatku sa

[1] V odbornej literatúre sa takýto typ systémov nazýva *sketch-based systems*

vysvetľuje rozdiel medzi *Dotazovacím režimom* a *Experimentálnym režimom* aplikácie. Jednotlivé časti aplikácie sú rozdelené do častí, ktoré sú prehľadne popísané.

Šiesta kapitola zahŕňa experimenty vykonané na implementovanom SBIR systéme. Bude popísané prostredie, na ktorom prebiehalo testovanie, samotný experiment a závery vyplývajúce z výsledkov. Čitateľ sa dozvie, na aký typ obrázkov je navrhnutý systém vhodný a ako správne zadávať dotaz pre dosiahnutie najlepších výsledkov.

Kapitola 2

Databázy digitálnych obrázkov

Pred vyhľadávaním obrázkov je treba si položiť otázku: ,,Čo chceme vyhľadať a akým spôsobom definovať dotaz?"V nasledujúcom texte bude vysvetlené, čo zahŕňa problém vyhľadávania obrázkov a čo všetko je potrebné si uvedomiť. Popísané budú CBIR systémy, význam detektoru a deskriptoru. Na záver bude popísaná jedna z metód detekcie hrán.

2.1 Vyhľadávanie obrázkov

Majme multimediálnu databázu, z ktorej by chcel užívateľ získať konkrétne obrázky. Aký druh dotazu má položiť databáze? Odpoveď na túto otázku vyžaduje detailne poznať potreby užívateľa – prečo hľadá obrázky, ako použije výsledok. V [5] sa uvádza, že obrázky sú potrebné z rôznych dôvodov, okrem iného aj pre:

- ilustráciu textu, ktorou vyjadríme informácie alebo pocity ťažko opísateľné slovami
- zobrazenie špeciálnych dát kvôli ich analýze (snímky z MRI[1])
- vytváranie formálnych nákresov pre neskoršie použitie (architektonické plány alebo nákresy elektrotechnických zapojení)

Získať požadovaný obrázok z databázy vyžaduje vyhľadanie obrázkov znázorňujúcich daný typ objektu alebo scény tak, aby mal užívateľ pocit, že výsledok je podobný dotazu. Obrázky obsahujú rôzne vlastnosti, ktoré je možné použiť pri vyhľadávaní ako napríklad:

- výskyt konkrétnej kombinácie farieb, textúr alebo významných tvarov (žltý kruh)
- výskyt alebo rozmiestnenie špecifických typov objektov (stoličky okolo stola)
- znázornenie konkrétneho druhu udalosti (hokejový zápas)

[1]Magnetic Resonance Imaging = Magnetická rezonancia

- výskyt známej osoby alebo špecifické miesto (prezident zdraví ľud)
- subjektívne emócie (scéna evokuje u osoby šťastie, radosť)
- metadáta, určujúce, kto vytvoril obrázok, kde a kedy

Jednotlivé vlastnosti by mohli byť využité pri definovaní typu dotazu do databázy. Každá nasledujúca vlastnosť v zozname (okrem poslednej) reprezentuje vyššiu úroveň abstrakcie ako predchádzajúca, a zároveň je aj ťažšie nájsť odpoveď na dotaz, ktorý vyhľadáva podľa danej vlastnosti. V [5] sa uvádza, že dotazy je možné rozdeliť do troch úrovní podľa rastúcej zložitosti odpovede. Jednotlivé úrovne popíšeme len v stručnosti.

Úroveň 1 zahŕňa vyhľadávanie podľa primitívnych vlastností ako farba, tvar, textúra alebo priestorové usporiadanie primitívnych elementov. Príkladom takéhoto typu dotazu môže byť: "Nájdi obrázky s dlhou tenkou čiarou v pravom hornom rohu!".

Úroveň 2 predstavuje vyhľadávanie podľa odvodených základných vlastností. Takýto druh dotazu vyžaduje istý typ logickej dedukcie a schopnosť identifikovať zobrazené objekty. Napríklad: "Nájdi obrázok dvojposchodového autobusu!".

Úroveň 3 zahŕňa vyhľadávanie podľa abstraktných vlastností vyžadujúce vysokú úroveň dedukcie a schopnosť posúdiť zmysel alebo význam objektu, prípadne celej scény. Môže to byť dotaz typu: "Nájdi obrázok vykresľujúci utrpenie!".

Vyhľadávajúci algoritmus, ktorý bude vysvetlený v neskorších kapitolách využíva najzákladnejšiu vlastnosť obrázku, a tou je intenzita. Na jej základe lokalizuje výrazné a významné línie, ktoré sú použité pri porovnávaní s dotazom. Dotazy, ktoré navrhovaný algoritmus prijíma na vstupe, radíme do úrovne 1.

2.2 CBIR systémy

CBIR systém je skrátený názov anglického termínu *Content-Based Image Retrieval System*. Tento pojem popisuje systémy, ktoré z veľkej množiny obrázkov automaticky extrahujú preddefinované vlastnosti. Vyhľadávanie obrázkov použitím ručne pridaných kľúčových slov (tzv. tagov) sa nepovažuje za CBIR.

Je potrebné si uvedomiť, že digitálne obrázky pozostávajú z poľa obrazových bodov, ktoré samy o sebe nedávajú zmysel. A preto je nutné extrahovať užitočné informácie zo surových dát. Vytvorí sa tak metainformácia o obrázku, ktorá sa potom použije pri porovnávaní s dotazom. V podstate môžeme hovoriť o automatickom indexovaní digitálnych obrázkov.

Oblasť výskumu a vývoja CBIR systémov je rozsiahla a problémy, ktoré rieši, sú úzko späté s problémami spracovania obrazu a počítačového videnia.

Najdôležitejšie problémy, s ktorými sa stretneme pri návrhu a implementácii CBIR systému sú:

- pochopenie potrieb užívateľa (aký druh digitálnych obrázkov použije a čo očakáva od výsledku)
- identifikovanie spôsobu popisovania obsahu v obrázku (to znamená určiť vhodné vlastnosti obrázku, ktoré sa použijú na vytvorenie popisu jeho obsahu)
- extrahovanie takýchto vlastností zo surových dát
- poskytnutie kompaktného úložného priestoru pre veľké databázy obrázkov
- dosiahnutie zhody medzi dotazom a uloženými obrázkami tak, aby zodpovedala ľudskému chápaniu podobnosti (obrázky, ktoré považuje systém za podobné, by mal aj užívateľ považovať za podobné)
- efektívne pristupovať k uloženým obrázkom a vyhľadať ich podľa obsahu
- poskytnutie užitočného rozhrania pre komunikáciu užívateľa so systémom

Počas návrhu nášho systému sme narazili na všetky uvedené problémy. V ďalšom texte sa k nim budeme nepriamo vracať.

Rôzne implementácie CBIR systémov dávajú užívateľovi možnosť získavať dáta rôznymi technikami dotazovania, pričom vždy záleží na potrebách užívateľa. Dotazovanie v CBIR systémoch je vlastne hľadanie podobností, tzn. porovnáva sa vstupný obrázok s obrázkom z databázy. Výsledkom je číslo, určujúce, ako veľmi sa obrázok z databázy podobá užívateľskému dotazu. Na základe tohto čísla sa vyberie x obrázkov s najväčšou podobnosťou k dotazu. Často sa stáva, že obrázky na prvých pozíciách neobsahujú to, čo užívateľ hľadá, pričom CBIR to považuje za veľmi podobné. Dôvodom je práve sémantická medzera, ktorá sa len veľmi ťažko prekonáva. Jedna z možností ako zabrániť, aby sa do výsledku opäť dostali nevhodné obrázky, je spätná väzba užívateľa. K danej dvojici dotaz–obrázok sa pridelí jeden z troch znakov:

- **positive** – vyhľadaný obrázok spĺňa požiadavky užívateľa
- **neutral** – užívateľ nedokáže rozhodnúť, či bol obrázok správne vyhľadaný
- **negative** – vyhľadaný obrázok nespĺňa požiadavky užívateľa, a teda bol nesprávne vybraný

Pri ďalšom vyhľadávaní tvoria pridelené príznaky akúsi váhu, ktorá môže rozhodnúť o konečnom usporiadaní obrázkov. Ale nie vždy je možné použiť tento prístup.

Spôsobov zadávania dotazu CBIR systému existuje niekoľko. Spomeňme iba:

- **query-by-image** – dotaz je obrázok alebo časť obrázku

- **query-by-sketch** – dotaz je načrtnutý (nakreslený) užívateľom
- **query-by-gestures** – dotaz užívateľ zadáva ako gestá pomocou ruky, viď [3].

Táto práca implementuje CBIR systém, do ktorého sa vstupný dotaz zadáva spôsobom *query-by-sketch*. Podobným architektúram sa tiež hovorí *Sketch-Based Image Retrieval Systems*, skrátene SBIR.

2.3 Detektor a deskriptor výrazných častí obrazu

Koncept detekovania výrazných častí obrazu[2] je založený na metóde, ktorá sa snaží vytvoriť abstraktnú reprezentáciu z obrazovej informácie. O každom bode obrázku rozhoduje, či patrí do jedného z možných druhov výraznej časti obrazu. Výsledkom je podmnožina z obrazovej domény, často vo forme izolovaných bodov, súvislých kriviek alebo spojitých regiónov [10].

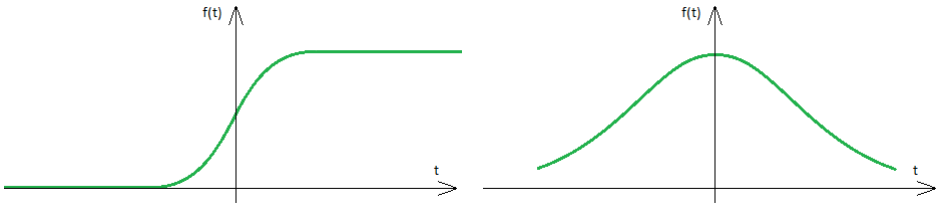
Neexistuje univerzálna definícia, čo je alebo z čoho pozostáva výrazná oblasť obrazu, predovšetkým z dôvodu že presná definícia vždy závisí na spôsobe aplikácie. Obecne sa ale dá povedať, že ide o zaujímavú oblasť, región alebo bod v obraze. Zaujímavosť spočíva napríklad vo farbe, intenzite svetla vzhľadom na svoje okolie a i. Existujú rôzne druhy výrazných častí obrazu, my sa zameriame na hrany.

Hrana je tvorená bodmi, ktoré ležia na hranici medzi dvoma rôzne farebnými regiónmi. Obecne môže hrana nadobúdať ľubovoľný tvar. Zvyčajne býva definovaná ako množina bodov, ktoré majú výrazný gradient, teda veľkosť gradientov je výrazne väčšia v okolí hranových bodov. Určením bodov, ktoré tvoria hranu, úloha detekcie hrany nekončí. Niektoré body môžu tvoriť šum alebo dokonca môžu ležať mimo skutočnú hranu. Existujú rôzne metódy, ktoré vyberú správne obrazové body, napríklad počítaním šírky hrany alebo určením jej zakrivenia. To znamená, že za pomoci rôznych obmedzení na vlastnostiach, ako je tvar hrany, jemnosť hrany alebo veľkosť gradientu, dokážu lepšie detekovať hranu.

Algoritmy ktoré hľadajú výrazné časti obrazu, sa nazývajú *detektory*. Žiaduca vlastnosť každého detektoru je nájdenie rovnakej výraznej časti na dvoch, respektíve viacerých obrázkoch podobnej scény. Takýto algoritmus nám dáva možnosť hľadať podobné obrázky.

Je potrebné si uvedomiť, že algoritmus pre rozpoznávanie obrazu je komplexný. Pozostáva z niekoľkých iných podalgoritmov, pričom každý má inú úlohu. Detektory tvoria jednu skupinu podalgoritmov a patria medzi tie, ktoré pracujú nad surovými dátami obrazu, to znamená, že ide o nízko-úrovňové operácie spracovania obrazu. A preto algoritmus pre rozpoznávanie obrazu

[2] V anglickej literatúre je tento pojem známy ako *image features*



Obr. 2.1: Funkcia vľavo reprezentuje signál intenzity pred použitím operátora gradientu. Vpravo je znázornená krivka prvej derivácie pôvodnej krivky vľavo.

bude len tak dobrý, ako dobrý je jeho detektor.

Ďalšou skupinou sú *deskriptory*, ktoré vytvárajú informáciu na vyššej úrovni z dát získaných z detektoru. Ich hlavnou úlohou je popisovať dáta získané z detektoru. Môžu popisovať komplexnejšie štruktúry z bodov alebo hrán, ako napríklad objekt. Objektom môže byť ľubovoľný útvar tvorený konkrétnym usporiadaním hrán. Nás budú zaujímať deskriptory, ktoré popisujú nejakú časť obrázku formou histogramu.

2.4 Metódy detekcie hrán

Obrazová scéna obsahuje veľké množstvo informácií. Vďaka detekcii a extrakcii významných hrán môžeme zo scény odfiltrovať pre naše účely nepotrebné informácie. Ostanú tak iba dáta, ktoré nesú informáciu o štruktúre obrazu.

Cieľom detekcie hrán je previesť obrázok na množinu kriviek, hrán. Hrany sa pokladajú za významné elementy scény. Tvoria hranicu medzi objektom a pozadím alebo iba vizuálne rozdeľujú dva rôzne farebné regióny, prípadne regióny s rôznou intenzitou svetla. Existuje viac užitočných algoritmov na detekciu hrán, obecne sa dajú rozdeliť na dve skupiny podľa toho, akým spôsobom hranu detekujú.

Prvá metóda využíva gradienty. Hľadá maximum a minimum v prvej derivácii intenzity obrázku, viď. [9]. Predstavme si, že súradnice x,y popisujú pozíciu v obrázku a na súradnici z je funkcia intenzity svetla z obrázku. Potom miesta, kde je rozdiel intenzít najväčší, sú označené ako hrana.

Majme nasledujúci signál viď. Obr. 2.1 vľavo. Potom na Obr. 2.1 vpravo pekne vidieť, že derivácia má maximum umiestnené v strede krivky pôvodného signálu. To, čo sme si názorne ukázali v jednorozmernom priestore, ide analogicky rozšíriť do dvojrozmerného priestoru. Teda gradienty na obrázku určíme na základe parciálnej derivácie v smere ôs x a y.

Keďže v počítačovej grafike sa vyžaduje, aby algoritmy pre spracovanie obrazu fungovali rýchlo, používa sa pre aproximáciu takzvaná konvolučná maska alebo tiež matica. Jej veľkosť je konštantná a podstatne menšia ako analyzovaný obraz. Konvolučná maska sa prikladá na každý bod obrázku, pričom sa vždy spočíta veľkosť gradientu.

Druhá metóda je založená na hľadaní miest v obrázku, kde je druhá derivácia intenzity rovná nule. Teda hovoríme o metóde Laplacian, a podobne

ako v prvom prípade, aj tu sa na aproximáciu používa konvolučná maska. Jej rozmer je 5 x 5 a je použitá pre druhú deriváciu v oboch smeroch. Pre naše potreby je vhodnejšia prvá metóda.

2.4.1 Sobelov detektor hrán

Sobelov detektor hrán je založený na metóde gradientov. Postup výpočtu gradientov je detailnejšie popísaný v [9], [7]. Na výpočet gradientov v smere osi x, G_x, sa používa konvolučná matica

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}$$

a v smere osi y, G_y, matica

$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

Veľkosť gradientu sa spočíta použitím rovnice $G = \sqrt{G_x^2 + G_y^2}$. Pre dosiahnutie väčšej rýchlosti výpočtu sa častokrát používa aproximácia v tvare $G = |G_x| + |G_y|$.

Podľa [7] si zadefinujeme uhol gradientu nasledovne

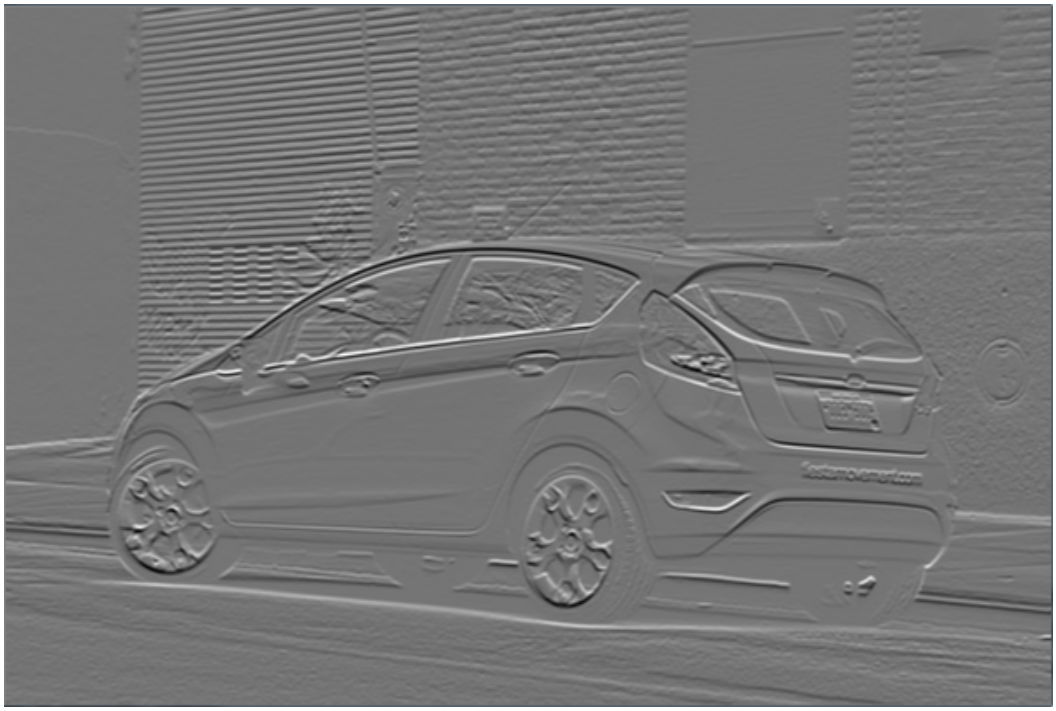
$$\Theta = \begin{cases} \arctan \frac{G_y}{G_x} & G_x \neq 0 \\ 0 & G_y = 0 \wedge G_x = 0 \\ \frac{\pi}{2} & \text{inak} \end{cases} \tag{2.1}$$

Na Obr. 2.2 je pôvodný obrázok, Obr. 2.3, 2.4 znázorňujú gradienty spočítané pomocou Sobela.

Okrem spomenutého Sobelovho detektoru existujú podobné varianty líšiace sa iba v použitej konvolučnej matici. Tie sa môžu líšiť ako v jednotlivých číselných konštantách v matici, tak veľkosťou matice. Podľa [2] platí, že väčšie konvolučné matice dávajú lepšiu aproximáciu, pretože menšie matice sú citlivé na šum. Je to spôsobené tým, že na výpočet gradientu sa využíva aproximácia a čím väčšia je matica, tým viac obrazových bodov pokryje, a teda tým lepšiu aproximáciu derivácie v danom bode dostaneme.

Obr. 2.2: Originálny obrázok v ktorom sa hľadajú gradienty.

Obr. 2.3: Vizualizácia gradientov spočítaných v smere osi x pomocou funkcie *cvSobel()* s konvolučnou maticou veľkosti 3.



Obr. 2.4: Vizualizácia gradientov spočítaných v smere osi y pomocou funkcie *cvSobel()* s konvolučnou maticou veľkosti 3.

Kapitola 3

Analýza a návrh

Hneď na začiatku kapitoly bude definovaný deskriptor pre popísanie lokálnej distribúcie hrán. Popíšeme algoritmus pre vyhľadávanie, ktorý bol navrhnutý podľa článku [6]. Následne budú vysvetlené navrhované zlepšenia. Zavedieme pojmy *posúvanie mriežky, veľmi podobné bunky* a *zašumené bunky.*

3.1 Prehľad

Vstupom vyhľadávajúceho mechanizmu je množina čiar, ktoré nakreslil užívateľ. Definuje tak tvar, ktorý chce vyhľadať. Výsledkom takéhoto dotazu je kolekcia obrázkov s podobným obsahom ako vstupný náčrtok.

Algoritmus na určovanie podobností je založený na základe deskriptorov, ktoré popisujú významný smer pre každú vymedzenú oblasť obrázku. Deskriptory sú spočítané pre každý obrázok databázy v offline procese. Pojem offline proces by sa dal v jednoduchosti vysvetliť ako množina všetkých výpočtov nevyhnutných pre beh databázy, ktoré treba vykonať na každom obrázku, prípadne na iných pomocných dátových štruktúrach, aby bolo možné databázu uviesť do stavu, keď od užívateľa očakáva vstup. Podľa definície CBIR systému v podkapitole 2.2 dochádza počas offline výpočtov k detekcii a extrakcii vlastností obrázku. To znamená, že sa automaticky vytvoria deskriptory pre každý obrázok patriaci databáze.

Počas behu aplikácie užívateľ nakreslením náčrtku vlastne poskytne pre každú vymedzenú oblasť obrázku smerovú informáciu. Vygenerovaný deskriptor sa potom jednoducho porovná voči všetkým deskriptorom v databáze.

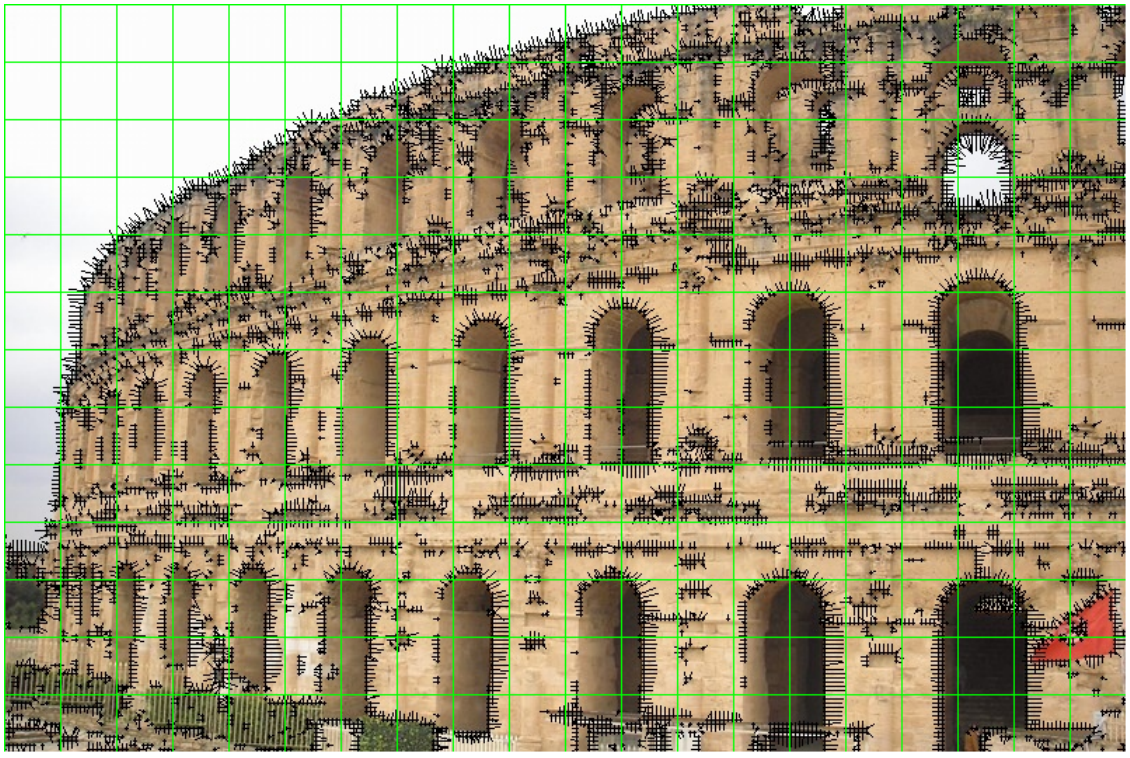
3.2 Deskriptor

Pojem deskriptor bol vysvetlený v podkapitole 2.3. V nasledujúcom texte bude podrobne popísaný deskriptor, ktorý bol predstavený v článku [6]. Je dostatočne robustný a zároveň kompaktný. Vzhľadom k tomu, že jeho pamäťové nároky sú malé, je vhodný na skutočne veľké databázy obrázkov. Taktiež samotný vyhľadávací algoritmus navrhnutý nad deskriptorom beží dostatočne rýchlo.

3.2.1 Dekompozícia obrázku na bunky

Majme obrázok I, ktorého rozmer je $m \times n$. Gradient v bode[1] (u,v) je definovaný výrazom $g_{uv} = \nabla I_{uv}$. Obrázok je pomyselne rozdelený na bunky rovnakej veľkosti, viď Obr. 3.1. Hovoríme, že $(u,v) \in C_{ij}$, ak pixel so súradnicou u a v je v bunke s indexom (i,j).

Hlavný význam deskriptorov popísaných ďalej v podkapitole 3.2.2 je určiť orientáciu veľkých gradientov v každej bunke obrázku s očakávaním, že budú korelovať s normálami na čiary nakreslené užívateľom. Dôležitý rozdiel medzi gradientmi užívateľského náčrtku od gradientov z obrázku je, že náčrtkové dosahujú podstatne väčšie rozmery. Je to zapríčinené rozdielom hodnôt farieb bielej a čiernej (v absolútnej hodnote to je 255). Bežné digitálne obrázky, s ktorými budeme pracovať, sú pestré na farby a prechody medzi farebnými regiónmi nie sú také výrazné. Preto veľkosti gradientov budú podstatne menšie. Okrem toho nás zaujíma hlavne percentuálne vyjadrenie, koľko gradientov je v danom smere. Preto výsledný deskriptor bunky musí byť normalizovaný.



Obr. 3.1: Obrázok je rozdelený na bunky. Malé čierne čiarky znázorňujú orientáciu a veľkosť gradientov.

[1]Bodom (u,v) sa myslí pixel obrázku na súradnici (u,v).

3.2.2 Deskriptor hranového histogramu

MPEG-7 je známy štandard vytvorený Medzinárodnou organizáciou pre normalizáciu (ISO). Jeho formálny názov je Multimedia Content Description Interface a bol popísaný v dokumente ISO/IEC 15938. Cieľom bolo vytvoriť štandard pre popisovanie multimediálneho obsahu. Jedna z ôsmych častí, do ktorých je štandard rozdelený, sa nazýva MPEG-7 Visual Standard a definuje deskriptory, ktoré môžu byť použité na meranie podobností medzi obrázkami alebo videami.

Deskriptor hranového histogramu (ďalej už len EHD[2]) je jedným z týchto deskriptorov. Bol navrhnutý tak, aby jeho veľkosť bola čo najviac kompaktná kvôli efektívnemu ukladaniu metadát. Jeho úlohou je popisovanie lokálnej distribúcie hrán.

Hrany tvoria významné prvky, ktoré reprezentujú obsah obrázku. Ľudské oko je citlivé práve na hrany z pohľadu vnímania obsahu obrázku. Jednou z možností, ako reprezentovať distribúciu hrán na obrázku, je použitie hranového histogramu.

Histogram je obecne najčastejšie používaná štruktúra pre potreby reprezentácie globálnych feature (tj. významných čŕt) v obrázku. Je invariantný voči posunutiu, otočeniu a normalizáciou ide dosiahnuť aj invariantnosť voči merítku (tj. zväčšenie alebo zmenšenie). Vďaka týmto vlastnostiam ide o vhodný nástroj, ktorý sa dá použiť napríklad na indexovanie obrázkov, ako je uvedené v článku [4]. Známy príklad použitia histogramu je určenie percentuálneho výskytu farieb na obrázku.

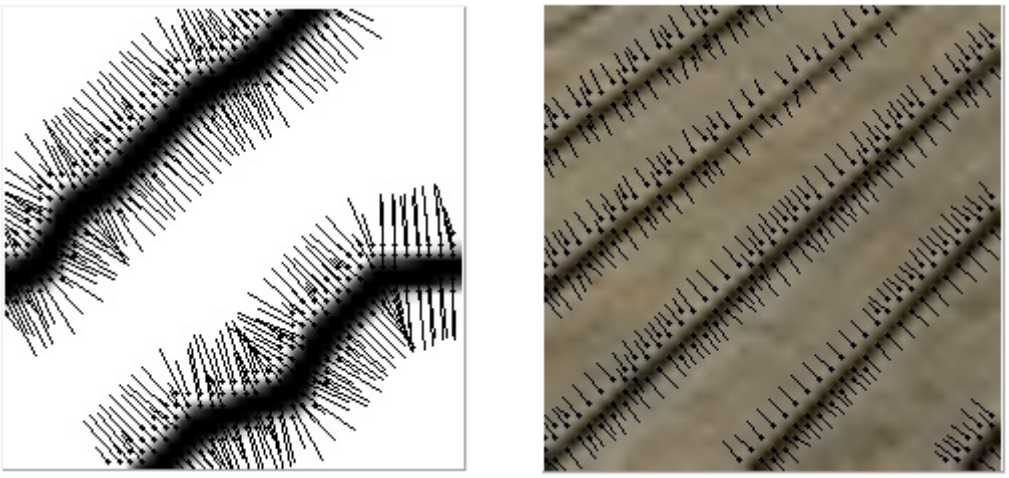
V článku [4] je popísané, akým spôsobom bol EHD definovaný podľa MPEG-7 štandardu, my sa ale nebudeme ďalej venovať tejto pôvodnej verzii a namiesto toho bude vysvetlené vylepšenie podľa článku [6].

Je zrejmé, že hlavná informácia, ktorú môže užívateľ prostredníctvom svojho náčrtku poskytnúť ako vstup, je smer čiary[3]. Tento smer je možné určiť pomocou normály. Smer gradientu v nejakom bode priamky je zhodný so smerom normály k priamke. A práve na tejto vlastnosti gradientu je založená upravená verzia EHD.

Pretože gradient je definovaný uhlom a smerom, v článku [4] je smer gradientu ignorovaný, lebo určuje len, ktorý z dvoch farebných regiónov má väčšiu intenzitu. Nás ale zaujíma, uhol a veľkosť gradientu v danom bode. Je jasné že na rozhraní dvoch farebne odlišných regiónov vzniká vizuálne čiara a práve tú treba deskriptorom popísať. O to sa postarajú gradienty, ktoré sú kolmé na rozhranie. Vznikajú na oboch stranách pričom majú opačný smer, viď. Obr. 3.2. Snaha je docieliť, že dva protismerné gradienty budú považované za rovnako orientované. To znamená, že gradient s uhlom α a gradient s uhlom $\alpha+180°$ sa považujú za rovnaké.

[2]V anglickom preklade Edge histogram descriptor.

[3]Pre jednoduchosť uvažujeme iba priamu čiaru, pretože rukou voľne nakreslená čiara či krivka sa dá rozdeliť na postupnosť menších rovných úsečiek.



Obr. 3.2: Dve zväčšené bunky znázorňujúce smery a veľkosti gradientov. Na obrázku vľavo je bunka z náčrtku, vpravo bunka z obrázku.

Aby sa zabránilo rušivým elementom v obrázku, ktoré vznikajú napríklad JPEG kompresiou alebo nekvalitným snímkom, v článku [6] stanovili, že gradienty s veľkosťou menšou ako 5% maximálnej možnej veľkosti budú ignorované. Veľkosť gradientu v bode (u,v) sa spočíta ako $\mathrm{g}_{uv}^T \mathrm{g}_{uv}$. Podmienku pre elimináciu šumu možno zhrnúť do nasledovného výrazu

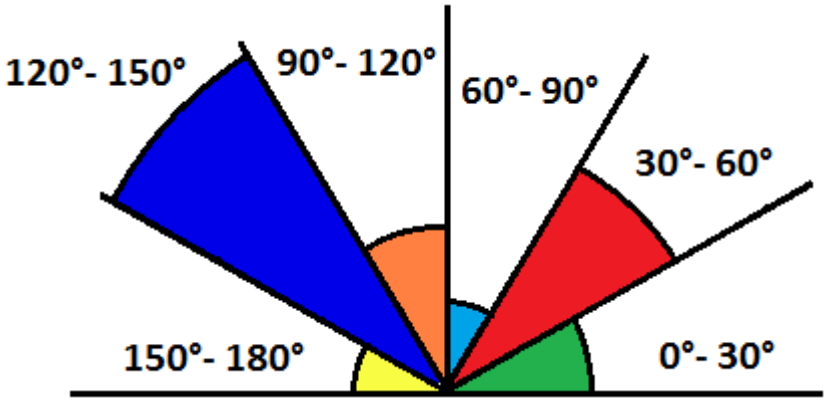
$$g_{uv}^T g_{uv} < \varepsilon^2. \tag{3.1}$$

kde $\varepsilon = \frac{\sqrt{2}}{20}$ (predpokladáme, že rozsah veľkostí x-ovej a y-ovej zložky gradientov je v intervale $\langle -1, 1 \rangle$). Gradienty sú spočítané s čierno-bieleho obrázku, ktorý vznikol z kanálu reprezentujúceho intenzitu svetla vo farebnom obrázku. V kapitole 6, venovanej experimentom, sa skúma, aký typ konvolučnej matice je najvhodnejší na výpočet gradientov.

Binárny obrázok sa skladá z pixelov nadobúdajúcich jednu z dvoch možných hodnôt. Typicky sa používa pre čiernu farbu hodnota 0 a bielu 255. U takéhoto typu obrázku sa jednoducho definuje, čo je objekt a čo je pozadie. Sémantický význam čierneho pixelu (pri zvolení dvojice čierna a biela) sa určí ako bod, ktorý patrí objektu a naopak biely pixel bude definovať pozadie obrázku.

Výhody binárnych obrázkov spočívajú v ich jednoduchej reprezentácii a tým vyplývajúcej možnosti rýchleho spracovania, respektíve získania informácií z nich. Používajú sa napríklad na identifikovanie obrysu objektu.

V článku [6] uvádzajú, že deskriptor náčrtku sa počíta priamo z jeho binárnej reprezentácie. Toto je práve jedna z vecí, ktorú sme v našej implementácii pozmenili. Náčrtok nereprezentujeme ako binárny obrázok, ale ako čierno-biely obrázok a jeho gradienty počítame tým istým mechanizmom ako gradienty obrázkov. Užívateľ má k dispozícii až štyri rôzne spôsoby kreslenia čiary. Ide o rôzne metódy renderovania kriviek na kresliacom plátne. Domnievame sa, že náčrtok, ktorý má krivky vyhladené (renderované za pomoci



Obr. 3.3: Vizuálne znázornenie histogramu, v ktorom sú gradienty rozdelené do 6 košov podľa ich smeru.

antialiasingu), bude lepšie pasovať na hľadaný obrázok ako náčrtok, ktorý má krivky ostré ("hranaté").

Gradienty bunky sú rozdelené do šiestich košov podľa toho, aký uhol zvierajú s osou x. Ako už bolo spomenuté, uhly α a $\alpha+180°$ sa považujú za rovnaké. Výsledná hodnota v danom koši je súčet druhých mocnín veľkostí gradientov, ktoré do neho patria. Druhou mocninou sa docieli, že väčšie gradienty sa zvýraznia a zároveň malé gradienty sa stanú menej významnými. Dôvodom zvýraznenia väčších gradientov je predpoklad, že užívateľ bude kresliť práve tie hrany (čiary), ktoré sú na obrázku výrazné. Výrazné gradienty sú zvýhodnené. Príklad deskriptoru jednej bunky je znázornený na Obr. 3.3.

3.3 Algoritmus vyhľadávania

V nasledujúcom texte budú požadované vlastnosti deskriptoru zapísané formou rovníc, z ktorých sa potom odvodí algoritmus pre vyhľadávanie. Táto podkapitola čerpá z článku [6].

Aby bol porovnávajúci algoritmus pre vyhľadávanie podobných obrázkov schopný povedať, ako veľmi sú dva obrázky podobné, potrebuje túto podobnosť nejakým spôsobom vyjadriť. Ako najefektívnejší a zároveň priamočiary prístup sa ponúka možnosť definovať podobnosť pomocou jedného čísla. Toto číslo nazvime *číslo podobnosti*. Podľa toho, akým spôsobom vyhodnocuje porovnávajúci algoritmus podobnosť, sa stanový, či menšia hodnota čísla znamená väčšiu podobnosť alebo naopak.

V tejto chvíli sa zadefinuje niekoľko rovníc, na základe ktorých funguje algoritmus pre spočítanie čísla podobnosti. Nech h_{ij} je histogram bunky C_{ij} s d gradientovými košmi, potom si veľkost k-teho koša definujeme ako

$$h_{ij}(k) = \sum_{(u,v)\in C_{ij}, o(g_{ij})\in[\frac{k}{d},\frac{k+1}{d}]} g_{uv}^T g_{uv} \tag{3.2}$$

pričom podľa článku [6] je

$$o(\mathbf{x}) = arccos(\frac{sgn(\mathbf{e}^T\mathbf{x})\mathbf{e}^T\mathbf{x}}{||\mathbf{x}||}) \tag{3.3}$$

kde $\mathbf{e}$ je ľubovoľný jednotkový vektor. Z rovnice je ale jasné, že obor hodnôt je v intervale $\langle 0, \pi/2\rangle$. Preto je nutné pridať ďalšie podmienky, aby sa obor hodnôt rozšíril na $\langle 0, \pi\rangle$.

Je zrejmé že histogram v bunke na náčrtku môže mať odlišný počet gradientov ako histogram bunky na obrázku, ktorý hľadáme. Z toho dôvodu je zavedený pojem normalizovaný histogram H_{ij} rovnicou

$$H_{ij} = \frac{1}{\sum_{x=1}^{k} h_{ij}(x)} h_{ij} \tag{3.4}$$

Medzi dvoma normalizovanými histogramami H_{ij} a $\tilde{H}_{ij}$ spočítame *vzdialenosť* L_1 ako

$$d_{ij} = \sum_{x=1}^{k} |H_{ij}(x) - \tilde{H}_{ij}(x)| \tag{3.5}$$

Platí, že *vzdialenosť* nadobúda hodnoty v intervale $\langle 0, 2\rangle$. Nula indikuje, že bunky sú vzájomne identické. Dva znamená, že sú maximálne rozdielne. Vzdialenosť medzi dvoma deskriptormi hranového histogramu H a $\tilde{H}$ je určená rovnicou

$$dist(H, \tilde{H}) = \sum_{i=1}^{q} \sum_{j=1}^{r} d_{ij} \tag{3.6}$$

a práve táto vzdialenosť určuje číslo podobnosti náčrtku a obrázku. Vo vzťahu sa nachádzajú dve premenné, q a r, ktorých hodnoty boli prednastavené na 20 $\times$ 13 (q je počet buniek obrázku na x-ovej osi a r je počet buniek obrázku na y-ovej osi). Rozmer mriežky by mal byť zvolený tak, aby sa bunky čo najviac podobali štvorcu. Najvhodnejšie rozmery mriežky sú predmetom skúmania v experimentálnej časti práce.

3.4 Vylepšený algoritmus vyhľadávania

Keďže výsledok vyhľadávania je závislí na kvalite vstupného náčrtku užívateľa, navrhnuté zlepšenia sa snažia túto závislosť zmenšiť. V nasledujúcom texte sú rozšírenia pôvodnej verzie vyhľadávajúceho algoritmu, navrhnutého v článku [6]. Všetky z nich boli implementované v programovej časti práce.

3.4.1 Gradienty

Ako už bolo spomenuté v podkapitole 2.4.1, použitím väčších konvolučných masiek je presnosť popisu derivácie intenzity väčšia. Z dôvodu overenia, ktorý typ masky je ideálny pre naše potreby, bola v programovej časti práce sprístupnená možnosť výberu konvolučnej matice rôznej veľkosti. Výsledky je možné nájsť v kapitole 6.

3.4.2 Prázdne bunky

Podľa článku [6] bunky z náčrtku, ktoré nemajú žiadne gradienty, sú počas porovnávania ignorované. Dôsledkom toho sa môže užívateľ zamerať iba na špecifický obsah obrázku a nemusí kresliť celý obrázok pred tým, ako položí dotaz. Zvýši sa tým množina akceptovateľných výsledkov a zároveň to výrazne obmedzí počet buniek, ktoré treba porovnať.

Zlepšenie, ktoré bolo implementované je nasledovné. Ak bunka z obrázku nemá gradienty, tak pri porovnávaní s náčrtkovou bunkou je ich vzdialenosť stanovená na 2, čo zodpovedá maximálnej odlišnosti. Tým sú znevýhodňované obrázky s prázdnymi bunkami.

Dôvod zlepšenia vychádza zo skutočnosti, že vzdialenosť medzi neprázdnou bunkou a prázdnou bude vždy 1. Obrázky s prázdnymi bunkami by tak mali väčšiu šancu sa dostať do výsledku a posunúť podobnejšie obrázky na nižšie pozície.

3.4.3 Posúvanie mriežky

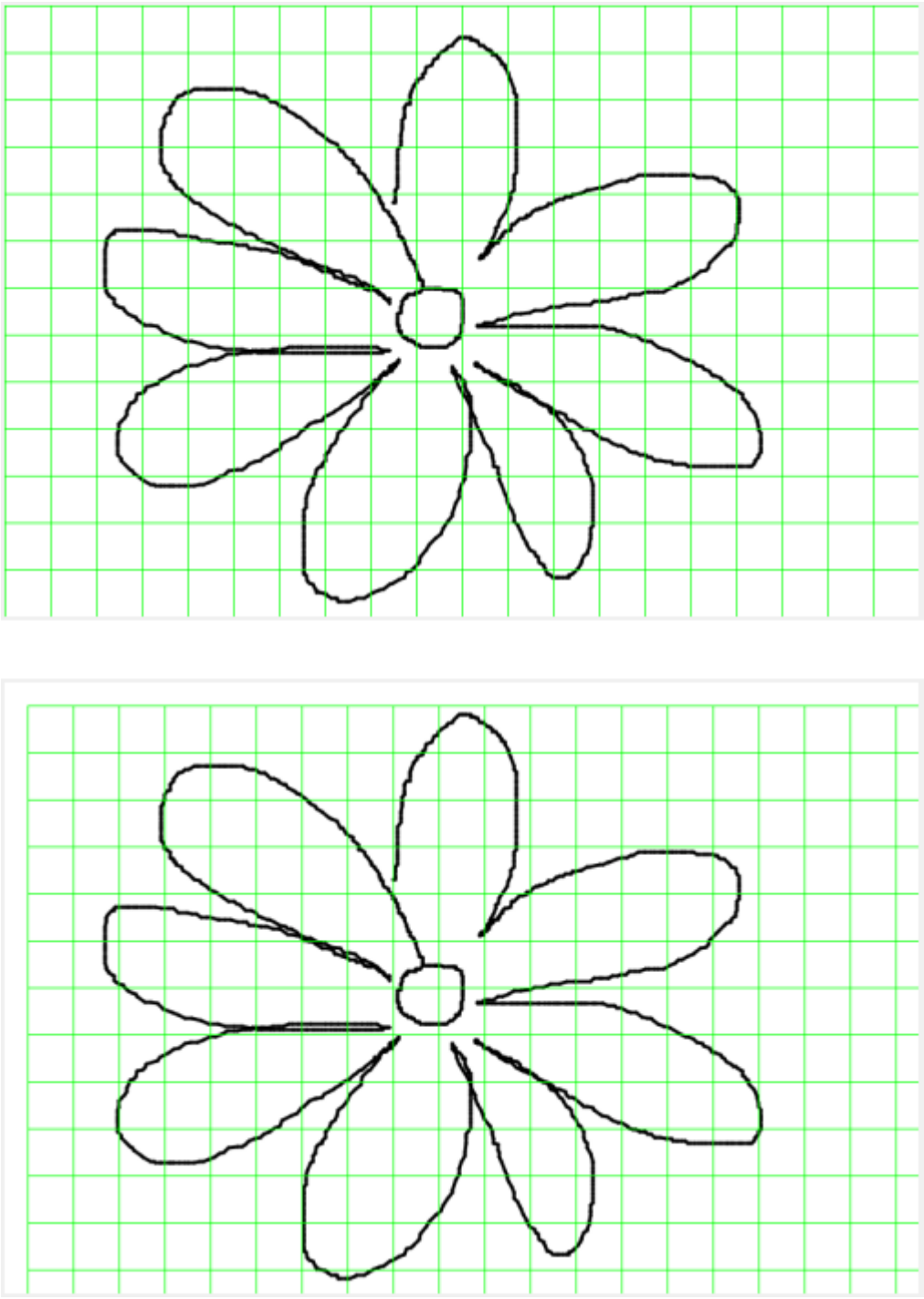
Je zrejmé, že nakreslením čiary voľnou rukou sa užívateľ nemusí trafiť do správnych buniek, čím dostane deskriptor, ktorý nezodpovedá dostatočne presne deskriptoru hľadaného obrázku. Obecne sa predpokladá, že užívateľ správne kreslí tvar objektu, až na jeho posunutie alebo veľkosť. Aby sa zabránilo chybám, vzniknutým počas kreslenia voľnou rukou, bol zavedený nový pojem, *posunutá mriežka*, ktorá má kompenzovať predovšetkým chybu v umiestnení. Snahou je teda upraviť navrhnutý deskriptor tak, aby bol invariantný voči posunutiu. Užívateľ ale musí mať možnosť túto invariantnosť obmedziť. Ako vedľajší efekt posúvacej mriežky je čiastočná invariantnosť voči merítku (veľkosť objektu). To ale vždy závisí od veľkosti použitej mriežky.

Na obrázku 3.4 sú znázornené dve rôzne mriežky. Prvá je pôvodná bez posunu, druhá je už s posunom na pozíciu (20, 20). Už na prvý pohľad vidieť, že v mriežke s posunom prechádzajú čiary cez iné bunky. Bunky navyše obsahujú iné gradienty, histogram bunky je odlišný rovnako ako aj celkový deskriptor obrázku.

Užívateľ má k dispozícii možnosť zadefinovať sám, aké mriežky sa použijú pri dotazovaní. Pre každú mriežku je potom spočítaná podobnosť s obrázkom. Finálna podobnosť medzi náčrtkom a obrázkom je daná najlepšie pasujúcou mriežkou (tj. vyberie sa najlepšie číslo podobnosti).

3.4.4 Veľmi podobné bunky

Ďalším návrhom na zlepšenie sa zavádza iný spôsob porovnávania, respektíve určovania podobnosti buniek. Vychádza sa z úvahy, že dve bunky sú vizuálne podobné, ak indexy košov s najväčšou hodnotou sa rovnajú a zároveň táto hodnota je aspoň 0.5 (tj. aspoň 50% gradientov patrí do košov s rovnakým indexom). Užívateľský náčrtok obsahuje len malé množstvo buniek spĺňajúce podmienky úvahy. Počet takýchto buniek sa dá využiť pre iné usporiadanie výsledkov.



Obr. 3.4: Horný obrázok s mriežkou na pozícii (0, 0). Spodný obrázok s mriežkou posunutou na pozíciu (20, 20)

O dvoch bunkách sa hovorí, že sú *veľmi podobné*, ak ich vzdialenosť je menšia ako *0.9*. Je zrejmé že táto definícia nesplňuje podmienky úvahy o vizuálnej podobnosti, ale pre praktické použitie je vhodná. Identifikovanie *veľmi podobných buniek* sa realizuje počas výpočtu vzdialenosti.

O každom obrázku sa dá potom povedať koľko *veľmi podobných buniek* obsahuje, čo je vyjadrené percentuálne vzhľadom k počtu buniek, ktoré boli porovnávané. Toto vyjadrenie sa potom používa k preusporiadaniu výsledkov dotazu. Očakáva sa, že obrázky, ktoré boli umiestnené na spodné priečky, sa vo výsledku dostanú na lepšie pozície.

3.4.5 Zašumené bunky a obrázky

Na Obr. 3.3 na prvý pohľad vidieť, že najväčšiu hodnotu obsahuje piaty kôš (modrý stĺpček). To znamená, že v danej bunke sú najvýraznejšie gradienty zvierajúce uhol s osou x v rozmedzí od 120° do 150°. Dal by sa urobiť záver, že touto bunkou pravdepodobne prechádza výrazná hrana, ktorá je určená gradientami z najlepšieho koša. Slovo pravdepodobne bolo použité úmyselne. Problém je v tom, že bunka môže obsahovať šum, chyby alebo veľa čiar, takže jej deskriptor bude podobný bunke, ktorou prechádza iba zopár nevýrazných hrán a jedna veľmi výrazná.

Súčet všetkých košov v normalizovanom histograme je vždy jedna. Ak vezmeme bunku, ktorej deskriptor má koše rovnomerne zaplnené (tj. nech každý kôš má hodnotu blízku 1 / 6 = 0.16), tak vo väčšine prípadov pri porovnaní s bunkou, ktorá ma plný iba jeden kôš, dostaneme

$$H_{ij} - \tilde{H}_{ij} = 1 - 1/6 = 0,83 \tag{3.7}$$

kde H_{ij} je bunka z náčrtku a $\tilde{H}_{ij}$ je bunka z rovnakej pozície z obrázku.

V podkapitole 3.4.4 bolo definované, že dve bunky, ktorých rozdiel je menší ako 0.9, sú veľmi podobné. *Zašumená bunka* je akoby univerzálna bunka, o ktorej porovnávajúci algoritmus vo väčšine prípadov prehlási, že je veľmi podobná s hocijakou inou bunkou, keďže ich rozdiel bude menší ako 0.9.

Cieľom je vedieť automaticky identifikovať *zašumené bunky* a obrázky, ktoré ich obsahujú, vyňať z výsledku dotazu, keďže také obrázky sa často krát dostávajú do výsledku. Niekoľkými experimentmi sa nám podarilo overiť, že ak každý kôš histogramu má hodnotu menej ako 0.25, tak bunku môžeme prehlásiť za *zašumenú*. Toto tvrdenie bolo overené experimentmi. Navyše, ak 15% zo všetkých buniek, ktoré majú gradient, sú zašumené, tak celý obrázok sa označí ako zašumený. Užívateľ má potom možnosť pred položením dotazu určiť, či sa pri vyhľadávaní použijú aj tieto obrázky.

Rozpoznávanie stromov

Identifikovanie zašumených buniek má ešte jeden prínos. Experimentálne bolo overené, že za pomoci zašumených buniek by sa dalo relatívne spoľahlivo rozpoznávať stromy, trávnaté porasty, listy alebo príroda obecne (viď Obr. 3.5). V kapitole 6 sa k tejto téme ešte vrátime.

Obr. 3.5: Zašumené bunky sú označené modrou farbou. Pomocou zašumených buniek sme dokázali rozpoznať a lokalizovať stromy na obrázku.

Kapitola 4

Implementácia

Kapitola sa venuje implementačnej časti práce. Budú spomenuté iba najdôležitejšie triedy, ktoré sú zodpovedné za logiku a beh vyhľadávajúceho algoritmu. Ide predovšetkým o triedy reprezentujúce dataset a deskriptor. Okrem toho sa zameriame aj na zaujímavé problémy, ktoré bolo potrebné riešiť. Napríklad problém s dlhými operáciami, ktoré trvajú desiatky minút.

Zdrojový kód bol dôkladne okomentovaný a obsahuje tak podrobný popis jednotlivých tried a metód.

4.1 Technológie

Programová časť práce bola implementovaná v jazyku C++ za pomoci vývojového prostredia Visual Studio 2008. Na prácu s digitálnymi obrázkami bola zvolená open source knižnica OpenCV. Ide o rozsiahlu knižnicu, ktorá obsahuje viac ako 500 optimalizovaných algoritmov, ktoré sú známe v oblasti počítačového videnia a spracovania obrazu. Pre naše potreby sme využili hlavne funkcie pre výpočet gradientov. Na podporu serializácie objektov sa použila knižnica *serialization*, ktorá je súčasťou rozsiahlej knižnice Boost. Na vytvorenie *user interface* sa používa cross-platformný framework Qt, [1].

Čo sa týka implementácie, z pohľadu zdrojového kódu je aplikácia rozdelená na dve vrstvy. Prvá tvorí užívateľské rozhranie a druhá algoritmy, deskriptor a dátové štruktúry potrebné pre samotný beh navrhovaného SBIR systému. Vďaka tomu je možné meniť užívateľské rozhranie bez väčších zásahov do najcitlivejších častí zdrojového kódu určeného pre SBIR systém.

4.2 Dataset

Predtým, ako užívateľ začne vyhľadávať obrázky vo svojej databáze, musí vytvoriť dataset, nad ktorým pracuje porovnávajúci algoritmus. Ide o vytvorenie akéhosi indexu nad digitálnymi obrázkami. *Dataset* je teda tvorený samotnou množinou obrázkov a metainformáciami extrahovanými z množiny týchto obrázkov. V texte sa bude pod pojmom databáza obrázkov rozumieť adresár, ktorý obsahuje iba súbory s obrázkami a to vo formáte JPEG.

4.2.1 Vytváranie datasetu

Vytváranie datasetu je časovo náročná operácia, ale stačí ju vykonať iba raz. Keďže databáza obrázkov môže nadobúdať veľké rozmery (v našom prípade najväčšia použitá databáza obsahuje 111 222 obrázkov), nie je vhodné uchovávať všetky metainformácie počas výpočtu v pamäti naraz, a preto sa priebežne ukladajú. Po každých 5 000 spracovaných obrázkoch sa uložia spočítané deskriptory do pomocných súborov *dumpFileX.aux*, kde X reprezentuje poradové číslo. Súbory sú ukladané v binárnom formáte kvôli rýchlemu načítaniu do pamäte a navyše sa ukladajú iba také informácie, ktoré nie je možné rýchlo dopočítať počas načítavania. Vďaka tomu je veľkosť súborov menšia.

Obecné informácie o datasete sa ukladajú do samostatného súboru *dataset.sbir*, ktorý je ukladaný v textovom formáte, pretože bolo cieľom umožniť skúsenému užívateľovi upravovať parametre obyčajným editovaním súboru. Navyše tento súbor spravidla obsahuje iba zopár jednoduchých dát, ktoré je možné relatívne rýchlo spracovať. Tým pádom tento formát nijak radikálne nespomaľuje rýchlosť načítavania datasetu.

Aby bola rýchlosť vytvárania čo najväčšia, je úloha spustená vo viacerých vláknach. Počet vlákien sa určuje dynamicky podľa počtu jadier procesora, ktorými disponuje hardware, na ktorom aplikácia beží. Úzkym hrdlom je práve zápis na disk, kedy dochádza k čakaniu vlákien až do uvoľnenia disku, pretože niektoré z vlákien disk práve využíva na zápis. Napriek tomu je rýchlosť vytvorenia datasetu znateľne vyššia pri použití viacerých vlákien, ako keby bolo použité iba jedno.

4.2.2 Načítanie datasetu

Na začiatku sa najprv načítajú obecné parametre, z ktorých sa potom pokračuje na ostatné súbory. Treba podotknúť, že dataset je celý v pamäti a je teda zrejmé že veľkosť datasetu, s ktorým dokáže aplikácia pracovať, je obmedzená veľkosťou dostupnej operačnej pamäte. Tento problém sme sa nesnažili nijak špeciálne riešiť, keďže v práci sa viac sústredíme na zlepšenie deskriptoru a algoritmu porovnávania. V súčasnom stave sme schopní pracovať s datasetom veľkosti 111 222 obrázkov na stroji s operačnou pamäťou 4GB.

Okrem iných možností, ako zväčšiť maximálnu veľkosť datasetu, je tu priestor na zlepšenie dátových štruktúr, ktoré reprezentujú EHD, ale refactoring by bol časovo náročný, a preto na túto alternatívu iba upozorňujeme.

4.2.3 Štruktúra súborov

Všetky súbory nového datasetu sa ukladajú do samostatného adresára. Jeho názov určuje užívateľ. Tento adresár sa vytvorí v adresári *dataset*. Ide o hlavný adresár všetkých datasetov vytvorených nad danou databázou obrázkov. Ak neexistuje, je automaticky vytvorený hneď vedľa adresára, ktorý je označený ako databáza obrázkov.

Hlavný súbor *dataset.sbir* je uložený ako textový súbor, jeho štruktúra je nasledovná:

- **xCells yCells** – dve kladné čísla udávajúce rozmery mriežky, počet buniek na osi x a počet buniek na osi y (nesmie sa editovať)
- **kernel** – kladné číslo, kód použitej konvolučnej matice pre výpočet gradientov, povolené hodnoty sú {-1, 1, 3, 5, 7} (nesmie sa editovať)
- **algorithm** – kladné číslo, kód hlavného porovnávajúceho algoritmu (vždy nastavená hodnota 0, v budúcnosti je možné dorobiť iné metódy porovnávania)
- **cellAlgorithm** – kladné číslo, kód algoritmu porovnávajúci dve konkrétne bunky, povolené hodnoty sú {0, 1}, definujú výber z comboboxu (editovateľné)
- **gridsNum** – kladné číslo, počet preddefinovaných mriežok, za týmto riadkom nasleduje daný počet dvojíc čísel (editovateľné, počet nasledujúcich dvojíc musí byť rovnaký ako toto číslo)
- **X Y** – dvojica celých čísel, mriežka s posunom na pozíciu (x,y) (editovateľné)
- **pathImg** – reťazec definujúci relatívnu cestu do adresára s obrázkami (editovateľný iba názov adresára, obsah adresára by mal zostať nezmenený)
- **pathAux** – reťazec, na každom ďalšom riadku až do konca súboru je relatívna cesta k súboru s uloženými EHDMaker (editovateľné iba názvy súborov, samotný obsah nesmie byť menený)

Súbory s koncovkou **.aux* sú binárne. Na začiatku súboru je vždy číslo určujúce počet serializovaných *EHDMaker* tried. Ich štruktúra je nasledovná:

- **imgName** – názov obrázku
- **noiseNumber** – kladné číslo, počet buniek ktoré sú považované za zašumené (pojem zašumená bunka bude vysvetlený neskôr)
- Nasledujúca štruktúra dát sa opakuje toľkokrát, na koľko buniek je obrázok rozdelený, hodnoty sú získané serializovaním EHD objektu:
  - **numGrad** – kladné číslo, počet gradientov v bunke. Ak je rovný nule, ďalej už nenasledujú žiadne dáta a pokračuje sa ďalším EHD objektom.
  - **sumAll** – kladné reálne číslo, súčet veľkostí gradientov pred ich normalizáciou
  - **A B C D E F** – kladné reálne čísla, hodnoty v košoch hranového histogramu

4.3 Implementácia SBIR systému

Spomenieme iba najdôležitejšie triedy, ktoré sú zodpovedné za logiku a beh algoritmu pre vyhľadávanie. Ide predovšetkým o triedy reprezentujúce dataset a deskriptor. Všetky patria do *namespace CBIR*.

4.3.1 Objektový model deskriptoru

UML diagram na obrázku 4.1 znázorňuje vzťahy medzi popisovanými triedami.

EHD

Základná stavebná jednotka mriežky je trieda *EHD* (Edge Histogram Descriptor). Obsahuje histogram, informácie o počte gradientov a celkovom súčte dĺžok gradientov v danej bunke. Histogram sa normalizuje po zaradení všetkých gradientov bunky do správnych košov. Trieda je navrhnutá tak, že jej údaje sa postupne plnia počas výpočtu. Nie je možné priamo nastaviť konkrétnu hodnotu koša.

Metódou *void add(float gu, float gv)* sa pridá gradient g_{ij} do histogramu (*gu* je x-ová zložka a *gv* y-ová zložka gradientu). Na zistenie správneho koša (tj. uhlu gradientu) sa používa metóda *void idx(float gu, float gv, float &magnitude, int &idx) const*, ktorá okrem indexu vracia aj veľkosť gradientu. Jeho kvadrát sa pripočíta k súčtu všetkých kvadrátov veľkostí v danom koši.

EHDMaker

Jednou z najvýznamnejších tried je *EHDMaker*, ktorá reprezentuje deskriptor jedného obrázku. Obsahuje cestu k obrázku, nad ktorým je vytvorená mriežka. Mriežka je reprezentovaná ako matica buniek EHD, na ktoré bol obrázok rozdelený. Rozdelenie obrázku je iba pomyselné. Bunky a ich histogramy sa počítajú v konštruktore triedy. Nasledujúce tri metódy sú zodpovedné za zostavenie objektu, ktorý reprezentuje plnohodnotný deskriptor obrázku:

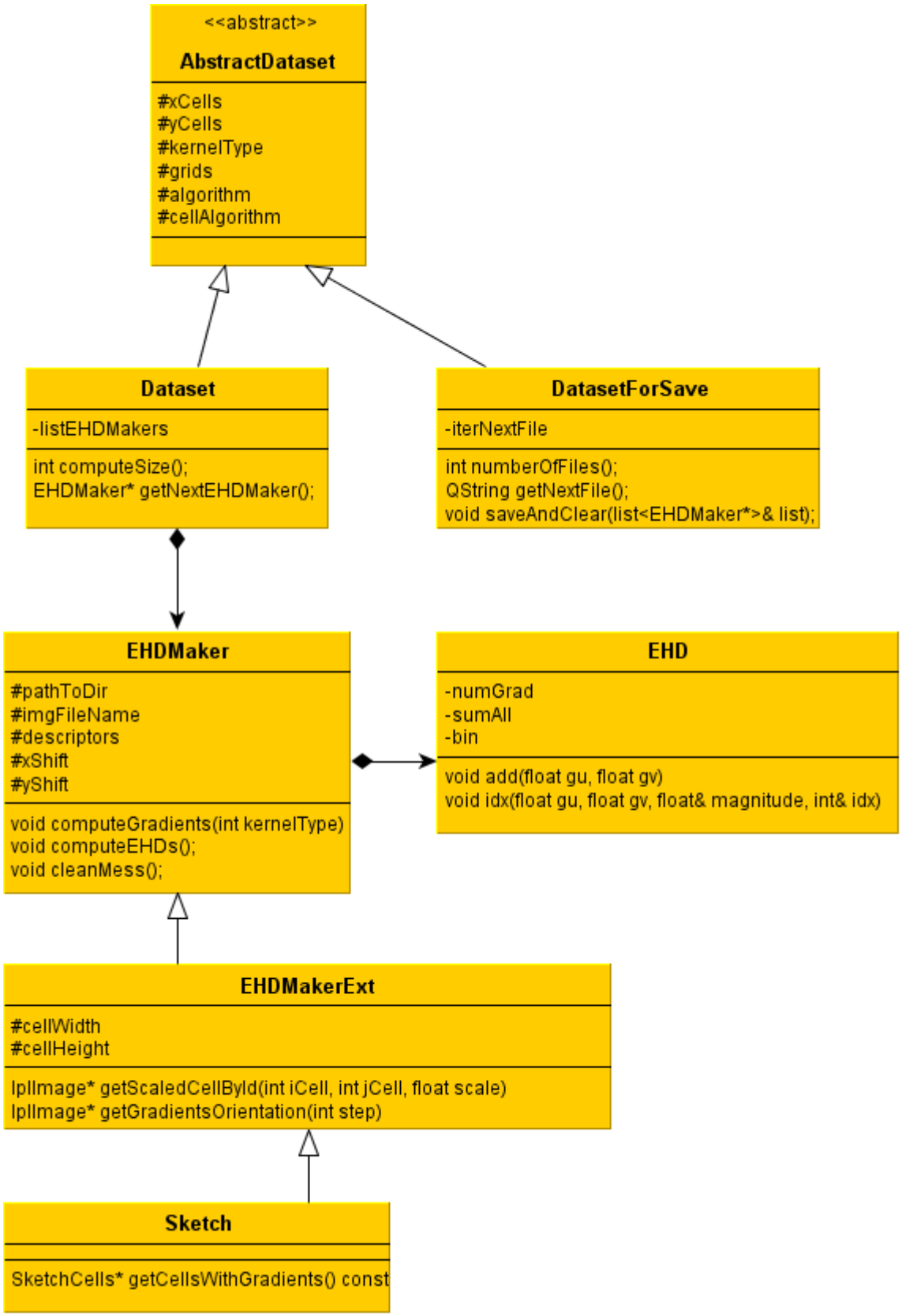
```
void computeGradients(int kernelType);
void computeEHDs();
void cleanMess();
```

Konštruktor volá metódy v tomto poradí. Ich úlohou je najprv spočítať gradienty podľa zvolenej konvolučnej matice. Z gradientov sa potom spočítajú histogramy jednotlivých buniek a na záver sa uvoľnia dáta potrebné len počas výpočtu. Na výpočet gradientov bola použitá funkcia *cvSobel()* z knižnice OpenCV, ktorá ako parameter dostáva veľkosť matice.

Pri návrhu triedy sa dbalo na to, aby pamäťové nároky na vzniknuté objekty boli čo najmenšie, pretože načítaný dataset v pamäti obsahuje práve tieto objekty.

EHDMakerExt

Priamy potomok triedy *EHDMaker* je *EHDMakerExt*. Ide o triedu, ktorá rozširuje predka o funkcie, ktoré užívateľovi poskytnú podrobné informácie



Obr. 4.1: UML diagram tried tvoriacich mechanizmus SBIR systému. Vymenované sú iba veľmi dôležité metódy a atribúty.

o uložených EHD, o gradientoch, z ktorých vznikli a i. Objekt tohto typu spravidla existuje v jednej chvíli iba jeden.

Spomeňme významnú metódu, ktorá sa používa na zobrazenie detailu bunky. Metóda *IplImage* getScaledCellByID(int iCell, int jCell, float scale)* dostáva ako parametre súradnice bunky a číslo definujúce, koľkokrát má byť originálna bunka zväčšená. Výsledkom volania je obrázok zväčšenej bunky s gradientami, ktoré boli pridané do jej histogramu. Vzhľadom k tomu, že užívateľské rozhranie pracuje s iným formátom obrázku ako knižnica OpenCV, bola zvolená konverzia medzi typmi obrázku prostredníctvom temporálneho súboru. Do súboru sa ukladá výsledný obrázok a pred zobrazovaním sa z tohto súboru načíta do správneho typu.

Ďalšia metóda *IplImage* getGradientsOrientation(int step)* vracia obrázok s gradientami. Vykreslené sú iba gradienty, ktorých veľkosť je aspoň 5% maximálnej veľkosti. Pomocou parametru sa udáva hustota vykreslených gradientov, t.j. ak $step == i$, vykreslí sa každý i–ty gradient. Je zrejmé, že nemá zmysel volať metódu s hodnotou parametru 1, keďže gradienty sú počítané v každom bode obrázku.

Sketch

Trieda *Sketch* reprezentuje náčrtok užívateľa a dedí od *EHDMakerExt*. Jej konštruktor ako jeden z parametrov dostáva obrázok, v ktorom je nakreslený dotaz. Taktiež sa zadáva posun mriežky, na základe ktorého sa vytvoria bunky EHD.

Pomocou metódy *SketchCells* getCellsWithGradients() const* je možné získať zoznam buniek, ktoré obsahujú gradienty. Podľa tohto zoznamu sa potom vie, ktoré obrázkové bunky treba porovnať.

4.3.2 Objektový model datasetu

Pre ukladanie a načítanie datasetu sa používajú dve rôzne triedy. Obe dedia od spoločného predka *AbstractDataset*.

DatasetForSave

Trieda je špeciálne navrhnutá pre ukladanie novovytvárajúceho datasetu. Metódou *int numberOfFiles()* sa načítajú názvy súborov, ktoré patria do databázy obrázkov. Zároveň sa vráti počet súborov databázy.

Iterátor *iterNextFile* ukazuje na aktuálny súbor čakajúci na spracovanie. Metóda *QString getNextFile()* vracia celú cestu ďalšieho súboru v poradí na spracovanie. Vrátený prázdny reťazec indikuje, že všetky súbory už boli spracované.

Počet obrázkov v databáze môže byť veľký. Obmedzenie veľkosti nepripadá do úvahy. Lenže uchovávať všetky spracované obrázky (tj. EHDMaker inštancie) v pamäti až do spracovania posledného obrázku je riskantné a zbytočne zaťažujúce stroj. V prípade, že by bol výpočet z nejakého dôvodu prerušený, prišli by sme o všetky výpočty. Preto je lepšie výsledky priebežne

ukladať. Pomocou metódy *void saveAndClear(list⟨EHDMaker∗⟩& list)* sa uloží zoznam výsledkov na disk a uvoľní sa pamäť, ktorú zoznam zaberal.

Dataset

Trieda funguje v podstate podobne ako *DatasetForSave*. Prechádza ale dvomi stavmi. V prvom sa načítajú dáta z disku (tj. vytvorenie inštancií EHDMaker) do zoznamu všetkých deskriptorov. V druhom stave je dataset pripravený na dotazovanie.

Prostredníctvom metódy *EHDMaker* getNextEHDMaker()* sa získavajú jednotlivé deskriptory obrázkov, ktoré je možné potom porovnávať s náčrtkovým deskriptorom.

4.4 Operácie

Niektoré operácie trvajú dlhšiu dobu. Aby mal užívateľ spätnú väzbu a prehľad, v akom stave je daná operácia, bol implementovaný obecný mechanizmus progress-barov. Trieda *AbstractOperation* je spoločný predok tried určených na spracovanie rozsiahlych úloh, ako je vytvorenie datasetu alebo načítanie datasetu. Každá trieda, implementujúca tohto predka, si sama povie, na koľko kúskov rozdeľuje rozsiahlu úlohu. Predok prostredníctvom hodín vie, kedy sa má spracovať ďalší kus úlohy. Medzi jednotlivými spracovaniami má dostatok času, aby aktualizoval progress-bar. Celý algoritmus je znázornený diagramom na Obr. 4.2.

Z diagramu sa dá vyčítať, že dôležitú úlohu tvoria hodiny, ktoré generujú pravidelné udalosti. V každom cykle sa aktualizuje stav progress-baru, ktorý sa prekresľuje, až keď sa hlavné vlákno aplikácie dostane ku slovu. Nasleduje spracovanie ďalšieho kroku. Ak išlo o posledný krok, hodiny sa zastavia a operácia končí. Inak sa celý proces opakuje.

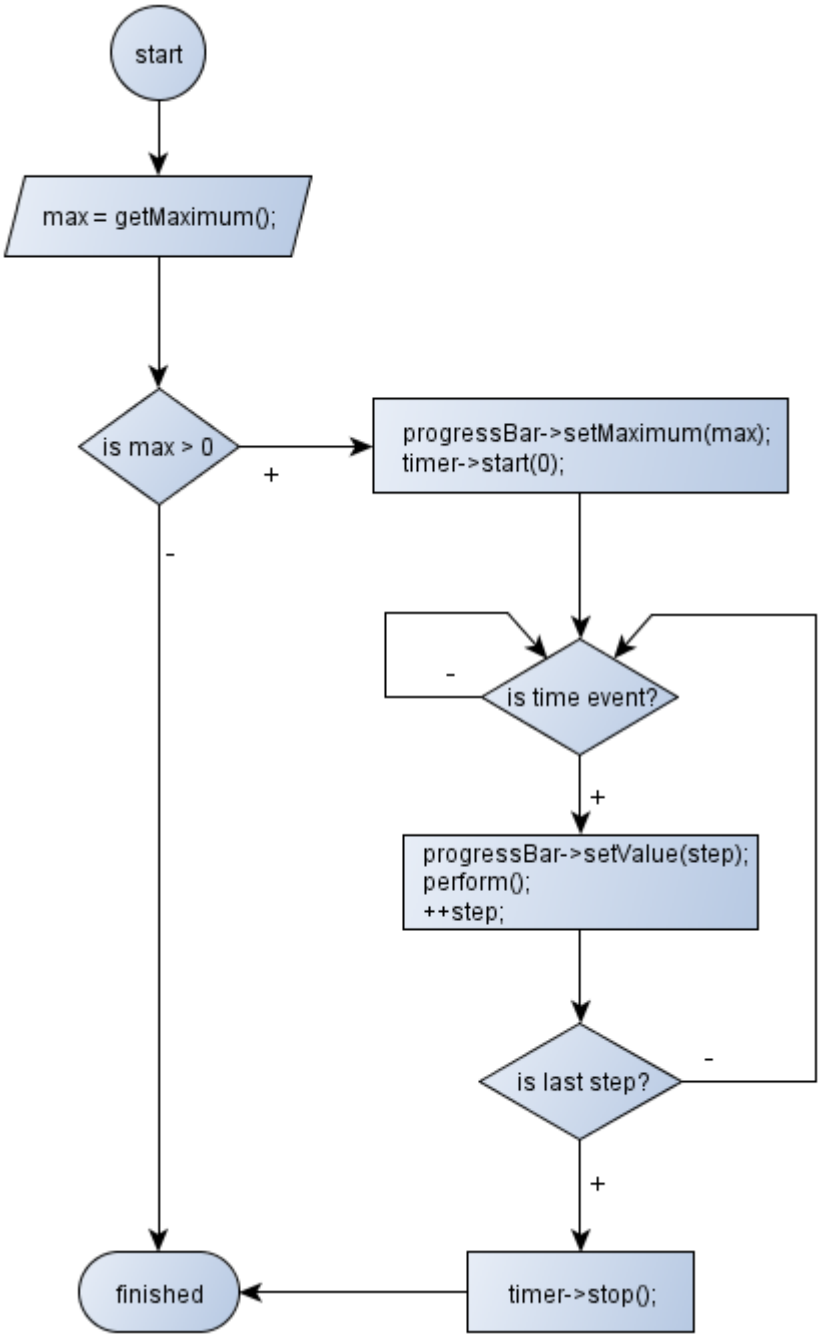
Teda pre každú úlohu, o ktorej sa predpokladá, že bude trvať dlhú dobu, je vytvorená špeciálna trieda *NázovÚlohyOperation*, ktorá je povinná implementovať metódy:

- *int getMaximum()* – vracia počet krokov potrebných na vykonanie celej úlohy. Napríklad načítanie datasetu by vrátilo počet obrázkov, ktoré treba spracovať.
- *void perform()* – v metóde sa implementuje samotné riešenie úlohy.

V nasledujúcich podkapitolách budú vysvetlené jednotlivé operácie, pričom niektoré využívajú na svoj beh vlákna.

4.4.1 Operácia vytvorenia datasetu

Operácia rozdeľuje metódu *perform()* do dvoch častí INIT a CREATE. Bol zvolený netradičný spôsob implementácie metódy *getMaximum()*, tá vracia hodnotu 2, pričom až v časti INIT sa dopočíta potrebný počet krokov. Ako už



Obr. 4.2: Diagram znázorňujúci algoritmus spracovania dlhých operácií.

bolo uvedené, počet častí, na ktoré je úloha rozdelená, sa zisťuje ešte pred spustením hodín. Vzhľadom k tomu, že operácia potrebuje najprv získať zoznam všetkých súborov na spracovanie a až potom začať spracovávať obrázky, bolo nutné vytvoriť INIT podúlohu, ktorá vykoná načítanie suborov. Je zrejmé, že pri veľkom počte súborov ich načítanie zaberie dlhšiu dobu. Počet súborov sa pripočíta k počtu krokov, ktoré treba vykonať. Následne prejde trieda do stavu CREATE a pri ďalšom volaní sa začnú spracovávať súbory obrázkov.

Ako už bolo spomenuté, operácia vytvárania datasetu, je optimalizovaná pre beh na viacerých vláknach. Pomocou Qt frameworku sa dá zistiť počet dostupných jadier procesoru nasledovne:

```
int coreCount = QThread::idealThreadCount();
if (coreCount == -1) {
    coreCount = 1;
}
```

Ak sa nepodarí identifikovať počet jadier, metóda *idealThreadCount()* z triedy *QThread* vracia hodnotu -1. V takom prípade bude operácia bežať iba v jednom vlákne.

Podľa počtu jadier sa vytvoria *QRunnable* úlohy, ktoré dostanú preddefinovný počet obrázkov na spracovanie. Tieto úlohy sa vhadzujú do *QThreadPool* inštancie, ktorá zabezpečí ich vykonanie. Keďže vlákna vzájomne zdieľajú spoločný zdroj, bolo potrebné ošetriť prístup k nemu technikou *Mutex* implementovanou frameworkom Qt.

4.4.2 Operácia načítania datasetu

Táto operácia beží iba v jednom vlákne. Kedže celú dobu číta z disku, nemá zmysel ju rozšíriť na viacvláknovú. Predpokladá sa, že dáta datasetu sú na jednom disku.

Metóda *getMaximum()* vracia počet obrázkov databázy a metóda *perform()* vykoná vždy jedno načítanie *EHDMaker* triedy.

4.4.3 Operácia vyhľadávania

Podobne ako operácia vytvárania datasetu, aj táto má metódu *perform()* rozdelenú do 5 častí: INIT, FIND, SORT, MODEL a CLEAN. Každá z týchto častí trvá dlhšiu dobu. Už z názvu vyplýva, k čomu sú jednotlivé časti určené. Predtým, ako sa začne vyhľadávať, je potrebné inicializovať náčrtky pre rôzne posuny mriežky. Následne sa začne prehľadávať, výsledok sa zotriedi[1], pripraví sa model do komponenty, ktorá je zodpovedná za ich zobrazenie. Na záver sa uvoľnia pomocné dáta.

Na riešenie časti FIND sú použité vlákna, aby bolo dotazovanie čo najrýchlejšie. Jak v predošlom prípade, tak aj v tomto je potrebné riešiť zamykanie. Zdieľaným zdrojom je zoznam obrázkov, ktoré tvoria databázu a s ktorými treba porovnať náčrtok.

[1]Keďže na testovanie sme mali k dispozícii databázu 111 222 obrázkov a zotriedenie poľa čísel takejto veľkosti nezaberá toľko času, nebolo potrebné uvažovať o inom spôsobe nájdenia k najlepších výsledkov.

Vytváranie modelu zahŕňa aj určenie počtu obrázkov zobrazených vo výsledku. Ten sa pre každý dotaz môže meniť. Minimálny počet obrázkov je 60 a maximálny 80. Ak pre niektorý obrázok v tomto intervale platí, že rozdiel čísla podobnosti nasledujúceho a práve kontrolovaného obrázku je väčší ako 0.15, ďalšie obrázky už nie sú pridané do výsledku.

Priestor na zrýchlenie tejto operácie nájdeme napríklad v spôsobe, akým sa triedia výsledky. Pre naše potreby je ale implementované riešenie dostatočné. V prípade použitia rozsiahlych databáz s počtom obrázkov okolo jedného miliónu bude nutné upraviť túto časť systému. Jedno z možných zlepšení je pamätať si počas výpočtu iba k najlepších výsledkov.

4.5 Kresliace plátno

Okrem kresliaceho plátna pre zadávanie dotazu (náčrtku) stojí za zmienku spomenúť aj komponentu pre zobrazovanie a skúmanie vybraných obrázkov. Obe majú niektoré vlastnosti a funkcionalitu rovnakú, a preto dedia od triedy *AbstractDisplay*. Tá rozširuje základnú Qt komponentu *QWidget*, vďaka ktorej je možné predefinovať metódy spracujúce udalosti myšky. Tie zabezpečia kreslenie čiary.

Pri navrhovaní komponenty sme sa inšpirovali konceptom MVC[2]. Modelom je dátová štruktúra reprezentujúca užívateľský náčrtok. View sa generuje pri udalostiach zobrazenia mriežky, nastavenia pozadia obrázku alebo zobrazenia gradientov obrázku z pozadia.

Pomocou frameworku Qt bolo jednoduchým spôsobom definované vykresľovanie náčrtku. V preťaženej metóde *void paintEvent(QPaintEvent* e)* sa definuje, ako sa má komponenta vykresliť:

```
void SketchCanvas::paintEvent(QPaintEvent* e) {
    QPainter painter(this);
    if (logoIsVisible) {
        logo->draw(painter);
    } else {
        painter.drawImage(QPoint(0,0), *imageView);
    }
}
```

Buď sa vykreslí ikonka s textom oznamujúcim, že komponenta slúži na vkladanie dotazu, alebo sa vykreslí aktuálny stav modelu. Ikonka sa vykresľuje spravidla vtedy, keď model neobsahuje žiadne dáta.

4.6 Nástroj pre experimentovanie

Do aplikácie bol zakomponovaný nástroj, pomocou ktorého môže užívateľ jednoducho experimentovať s rôznymi parametrami datasetu a sledovať, ako jednotlivé nastavenia ovplyvňujú kvalitu vyhľadávania. Tento nástroj je implementovaný v dvoch triedach *TunerDialog* a *TunerExecutor*. Prvá vytvára užívateľské rozhranie. V užívateľskej dokumentácii je popísané celé dialógové okno.

[2]Model-View-Controller

Dialógové okno nepoužíva klasický dataset, keďže jeho životný cyklus je odlišný od toho, čo požadujeme. Z toho dôvodu bola vytvorená trieda *TunerExecutor*. Obsahuje všetky parametre, ktoré je možné meniť, a ktoré ovplyvňujú dataset. Po zadaní dotazu, parametrov datasetu a vybraní malej množiny obrázkov sa spustením vyhľadávania volá metóda *void TunerExecutor::run(IplImage* sketchImg, QList⟨ImageValue∗⟩& listImageValue)*, ktorá musí najprv vytvoriť deskriptory z vybranej množiny obrázkov na základe nastavených parametrov. Táto operácia je časovo náročná, preto by množina testovaných obrázkov nemala presahovať počet viac ako 20. Avšak žiadne obmedzenia neboli implementované, je ponechané na užívateľa, koľko obrázkov si zvolí.

Nastavenia *TunerExecutor*-u je možné uložiť na disk a kedykoľvek ich použiť pri vytváraní datasetu.

Kapitola 5

Užívateľská dokumentácia

Táto kapitola sa venuje podrobnému popisu práce s aplikáciou. Bude vysvetlené, ako sa vytvára a používa dataset, akým spôsobom užívateľ zadáva dotaz, čo môže robiť s výsledkami. Taktiež si detailne rozoberieme takzvaný *experimentálny režim* aplikácie, jeho význam a funkcie.

5.1 Dotazovací a experimentálny režim

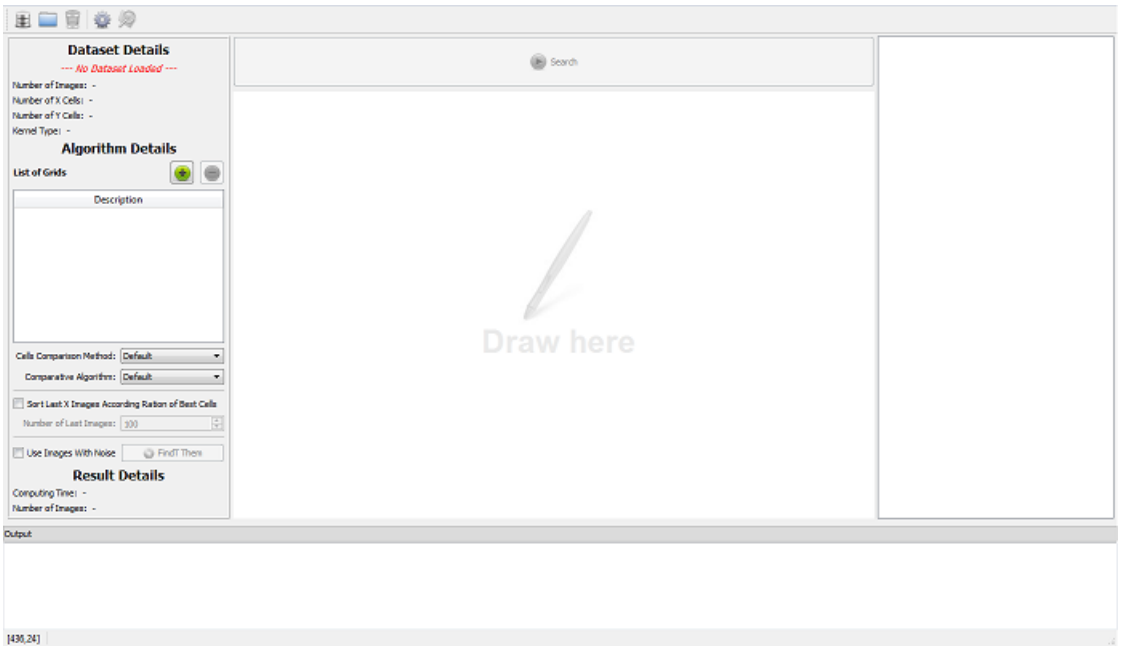
Predtým, ako začneme podrobne opisovať jednotlivé komponenty a ich funkčnosť, je potrebné si uvedomiť, že aplikácia pracuje v dvoch režimoch.

Dotazovací režim je k dispozícii hneď po štarte aplikácie. Najčastejšie je používaný pre načítanie datasetu obrázkov, nastavenie vlastností vyhľadávajúceho algoritmu, zadanie dotazu (náčrtku) a spustenie vyhľadávania. Užívateľ má potom možnosť prezrieť si výsledky dotazu, prípadne overiť správnosť výsledkov zobrazením gradientov alebo posúvaním mriežky. V tomto režime sa taktiež vytvárajú nové datasety.

Použité dátové štruktúry a pomocné algoritmy sú optimalizované tak, aby bol užívateľský dotaz čo najrýchlejšie spracovaný, a zároveň aby samotný dataset v pamäti zaberal čo najmenej miesta.

Experimentálny režim nie je vhodné používať na veľkú množinu obrázkov, vzhľadom k tomu, že použité dátové štruktúry, reprezentujúce deskriptory extrahované z obrázkov, obsahujú detailnejšie informácie, ktoré je možné v tomto režime skúmať a testovať. Vytvorenie týchto informácií zaberie nejaký čas a ich samotné uchovanie zase priestor v pamäti. Hlavný význam režimu tkvie v možnosti rýchlo otestovať rôzne parametre datasetu, ktoré sa dajú potom použiť pri vytváraní veľkých datasetov alebo skúmať výsledky dotazu získané v *Dotazovacom režime.*

Do experimentálneho režimu sa z hlavného okna dostaneme dvoma spôsobmi. Vzájomne sa líšia iba v tom, či hneď po prechode máme k dispozícii testovacie dáta alebo nie. Užívateľ potom prostredníctvom rozhranie sleduje detailné in-



Obr. 5.1: Hlavné okno aplikácie

formácie o bunkách na konkrétnych súradniciach alebo môže kontrolovať, ako vyhľadávajúci algoritmus spočítal podobnosť medzi obrázkom a náčrtkom.

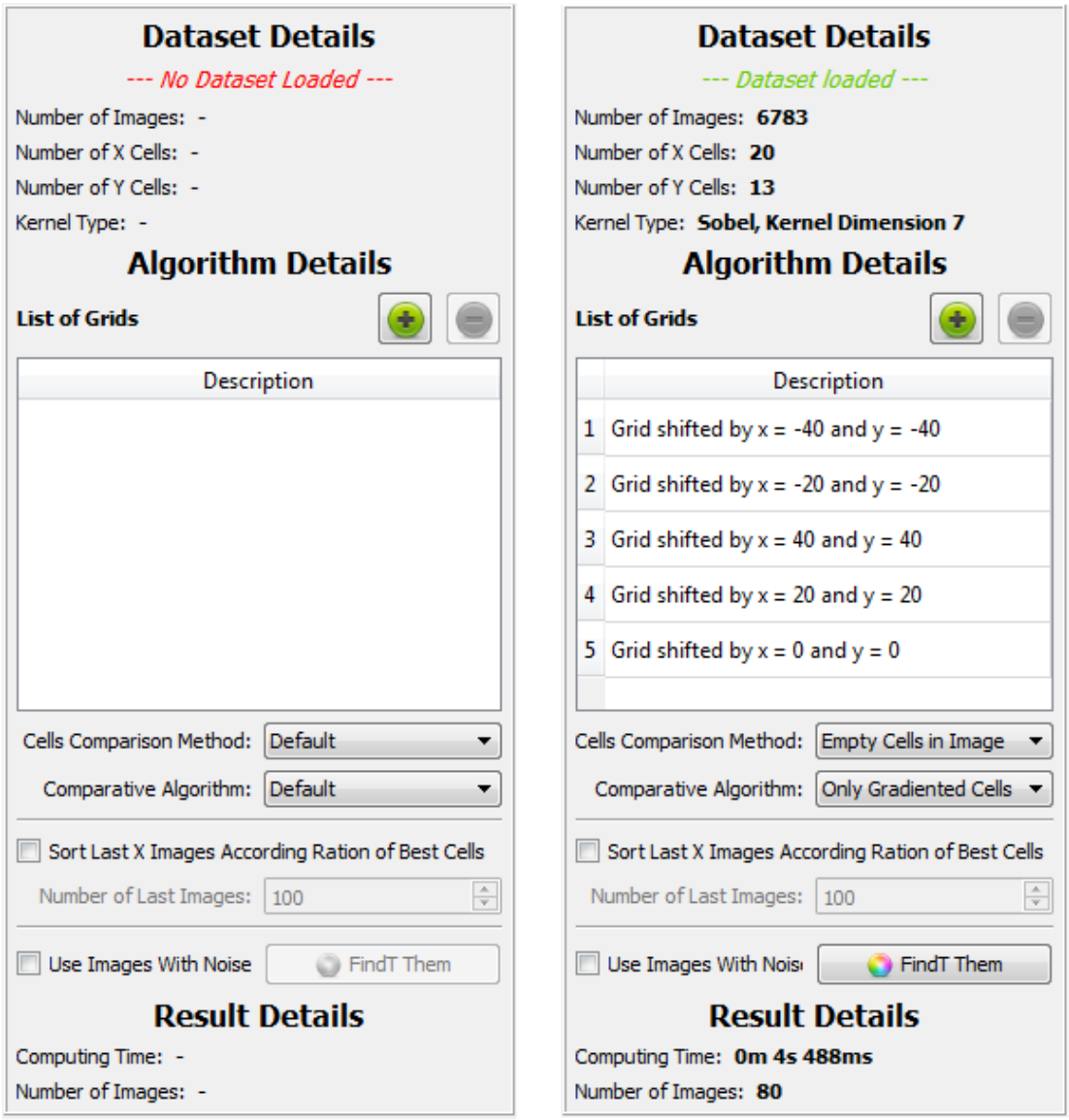
5.2 Hlavné okno aplikácie

Hlavné okno aplikácie, tak ako ho vidíme na Obr. 5.1, pozostáva z piatich častí. Postupne si vysvetlíme význam každej z nich.

Horná lišta obsahuje postupne tlačítka pre vytvorenie nového datasetu, načítanie datasetu do pamäte, uvoľnenie datasetu z pamäte a dve tlačítka, ktoré otvárajú dialógové okno v experimentálnom režime. Prvé z nich otvorí okno s preddefinovanými hodnotami. Druhé je prístupné práve vtedy, keď je v pamäti načítaný dataset. Pomocou neho je možné si do experimentálneho režimu preniesť užívateľom vybrané obrázky z panelu pre výsledky. Okrem toho sa vždy automaticky prenáša aktuálny dotaz (náčrtok), parametre datasetu a vyhľadávajúceho algoritmu.

Ľavý panel hlavného okna prechádza dvoma stavmi, ktorých vizuálnu odlišnosť vidieť na Obr. 5.2. Obrázok vľavo znázorňuje stav, keď v pamäti nie je načítaný žiadny dataset a preto nie je ani možné položiť dotaz (vtedy je tlačítko *Search* v strednom paneli Obr. 5.4 neprístupné). Vpravo je znázornená situácia, ako by mohol panel vyzerať vo chvíli, keď má v pamäti dataset.

Jedna z úloh panelu je teda zobrazovať informácie o počte obrázkov, z ktorých bol dataset vytvorený, ďalej rozmery mriežky a typ konvolučnej matice použitej pri výpočte gradientov. Toto všetko sú údaje, ktoré nie je možné



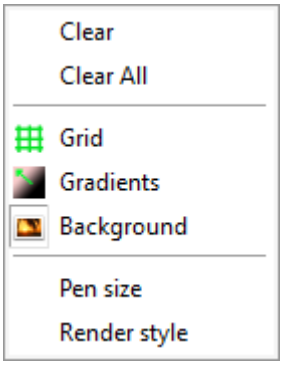
Obr. 5.2: Ľavý panel hlavného okna. Vľavo na obrázku je znázornený stav bez načítaného datasetu, v pravom obrázku vidieť údaje z načítaného datasetu.

editovať a sú pevne zviazané s datasetom. Naopak údaje v časti *Algorithm Details* sú určené k editácii. Užívateľ má možnosť meniť vlastnosti algoritmu pre vyhľadávanie, a tak ovplyvniť, aké obrázky sa zobrazia vo výsledku. Zadať môže rôzne posuny mriežky podľa toho, ako sám uzná za vhodné. Môže nastaviť spôsob, akým sa porovnávajú jednotlivé bunky. Podporované sú dve voľby, *Default*, ktorá implementuje algoritmus podľa článku [6] a *Empty Cells in Image*, ktorá znevýhodňuje bunky bez gradientov v obrázku datasetu.

Zaškrtávacie políčko *Sort Last X Images According Ration of Best Cells* je defaultne nezaškrtnuté, čo indikuje, že obrázky sú do výsledku triedené podľa *čísla podobnosti*, ktoré sú zadefinované v podkapitole 3.3. Zaškrtnutím sa mení spôsob triedenia a to nasledovne: najprv sa obrázky triedia podľa *čísla podobnosti* a na záver sa na posledných X obrázkoch spraví pretriedenie[1] podľa pomeru výskytu *veľmi podobných buniek*. Význam takýchto buniek bol popísaný v časti 3.4.4. Hodnota X sa nastavuje pomocou poľa *Number of Last Images*. Vhodné hodnoty sú v rozpätí od 100 do 250, ale žiadne takéto obmedzenie nebolo implementované.

Ďalšia významná vlastnosť, ktorá ovplyvní výsledok dotazu, je použitie obrázkov, ktoré by sme mohli nazvať zašumenými. Definíciu zašumeného obrázku sme uviedli v časti 3.4.5. Pomocou zaškrtávacieho políčka *Use Images With Noise* sa definuje, či algoritmus porovnáva náčrtok aj so zašumenými obrázkami alebo ich preskakuje. Pre predstavu, ako vyzerajú také obrázky, má užívateľ možnosť zobraziť len niekoľko najzašumenejších kliknutím na tlačítko *Find Them*.

V dolnej časti panelu *Result Details* sa nachádzajú informácie o trvaní posledného dotazu a počet obrázkov zobrazených vo výsledku.



Obr. 5.3: Kontextové menu, ktoré sa zobrazí po kliknutí pravým tlačítkom myšky na plochu náčrtku.

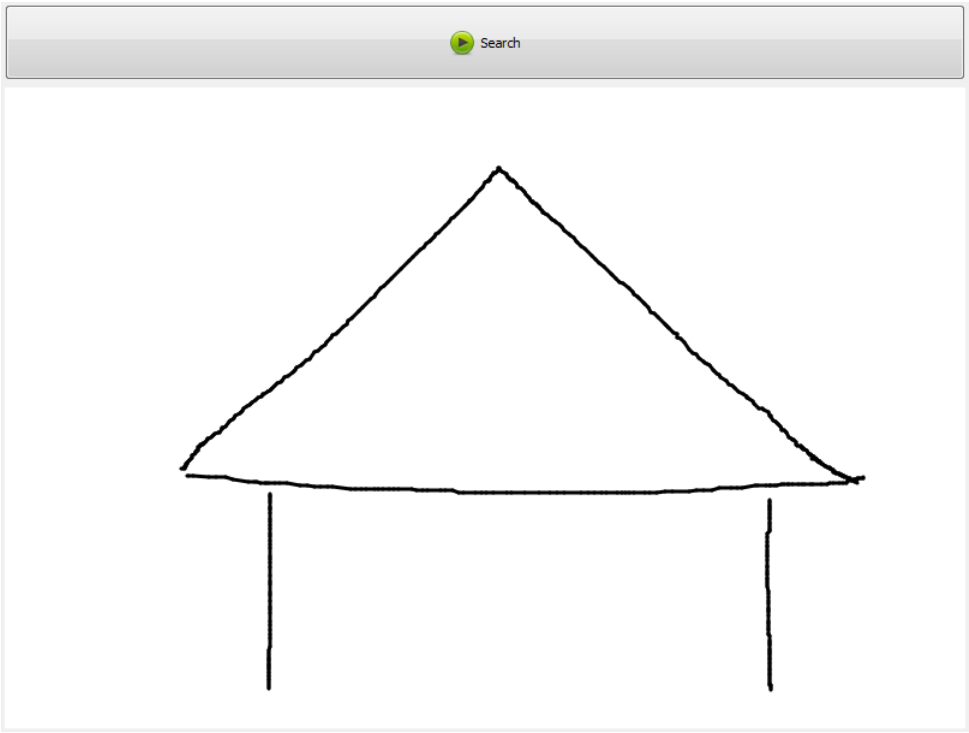
Stredný panel slúži na zadanie vstupného dotazu. Užívateľ pridržaním ľavého tlačítka myšky kreslí čiaru v smere, ktorým pohybuje myškou. Za predpokladu, že je v pamäti načítaný dataset a zároveň je zadaná aspoň jedna mriežka, môže užívateľ kliknutím na tlačítko *Search* spustiť vyhľadávanie podobných

[1] Tiež známe pod pojmom *re-ranking*

obrázkov. Detail panelu aj so zadaným ukážkovým dotazom je na Obr. 5.4.

Kresliaca plocha pre náčrtok slúži aj pre zobrazenie obrázku z výsledného listu. Užívateľ tak môže porovnať, ako na seba pasujú náčrtok a výsledok, prípadne môže použiť obrázok na pozadí ako predlohu, podľa ktorého vytvorí náčrtok. Ako už bolo spomenuté, dotaz by mal obsahovať významné línie, ktoré obsahuje aj hľadaný obrázok. Aby si užívateľ mohol overiť, či jeho náčrtok neobsahuje zbytočné čiary, ktoré by mohli pokaziť výsledok, môže si k obrázku na pozadí zobraziť gradienty definujúce dôležité línie.

Po kliknutí pravým tlačítkom na kresliacu plochu sa zobrazí kontextové menu, Obr. 5.3. Prvý príkaz zmaže iba náčrtok, pričom pozadie, mriežka a gradienty ostanú nezmenené. Druhý príkaz zmaže celú kresliacu plochu, skryje pozadie s gradientami a mriežku. Ďalej, príkaz *Grid* zobrazí alebo skryje mriežku. Príkaz *Gradients* je prístupný, iba ak je práve nastavené pozadie. Zobrazuje alebo skrýva gradienty obrázku. Príkaz *Background* skryje pozadie aj s gradientami, ak boli viditeľné. Posledné dve voľby menia veľkosť hrotu kresliaceho pera a spôsob, akým sa renderuje kreslená čiara. Obe voľby ovplyvňujú výsledok dotazu.



Obr. 5.4: Plocha pre zadanie dotazu (náčrtku).

Pravý panel zobrazuje výsledky dotazu. Na Obr. 5.5 je vidieť výsledok najviac podobných obrázkov k náčrtku z Obr. 5.4. Kliknutím ľavého tlačítka myšky môžeme konkrétnu miniatúru obrázka nastaviť ako pozadie do stredného panelu. Okrem toho je možné vybrať niekoľko miniatúr naraz a preniesť ich do

experimentálneho režimu, kde sa dajú zobraziť detailné informácie o mriežke, bunkách a samotnom porovnávaní s náčrtkom.

Obr. 5.5: Panel s výsledkami vyhľadávania.

Output panel zobrazuje rôzne pomocné informácie. Spomeňme napríklad trvanie rôznych operácii (načítanie datasetu, vyhľadávanie), priebežné informácie o vytváraní datasetu, informácie o počte jadier dostupných na počítači a ďalšie.

5.3 Vytvorenie a načítanie datasetu

V podkapitole 4.2 bolo vysvetlené, čo je dataset, ako sa ukladá a ako vyzerá hierarchia adresárov, v ktorej sa držia uložené súbory. Prostredníctvom dialógového okna na Obr.5.6 užívateľ zadáva parametre nového datasetu.

Do poľa *Directory of Images* zadá celú cestu k databáze s obrázkami (pripomenieme, že ide o adresár, ktorý obsahuje iba obrázky vo formáte JPEG a ich rozmery sú 800×533). Do poľa *Dataset Name* zadá výstižný názov datasetu. Tento názov sa potom použije pri vytvorení rovnomenného adresára v hlavnom adresári všetkých datasetov (tj. *dataset*), ktoré boli vytvorené z danej databázy obrázkov.

Ďalej nasledujú parametre datasetu, na základe ktorých sa vytvoria EHD – tj. rozmery mriežky a veľkosť konvolučnej matice na výpočet gradientov. Tieto hodnoty jednoznačne špecifikujú typ datasetu a nie je možné ich už meniť po načítaní vytvoreného datasetu do pamäte. Naopak, údaje v paneli *Algorithm Properties* je možné meniť ľubovoľne pred každým novým položeným dotazom. Nemajú žiadny vplyv na vlastnosť vytvoreného datasetu a boli pridané do tohto okna, aby užívateľ mohol hneď po načítaní datasetu pracovať a vytvárať dotazy. Dajú sa tu definovať nové posuny mriežky alebo rôzne porovnávajúce algoritmy.

Tlačítko *OK* sa sprístupní až vo chvíli, keď je zadaná cesta k databáze obrázkov a názov datasetu ešte pre danú databázu nebol použitý.

Pomocou tlačítka znázorňujúceho disketu je možné načítať spomenuté parametre zo súboru, do ktorého boli tieto hodnoty uložené pri práci s dialógom v experimentálnom režime.

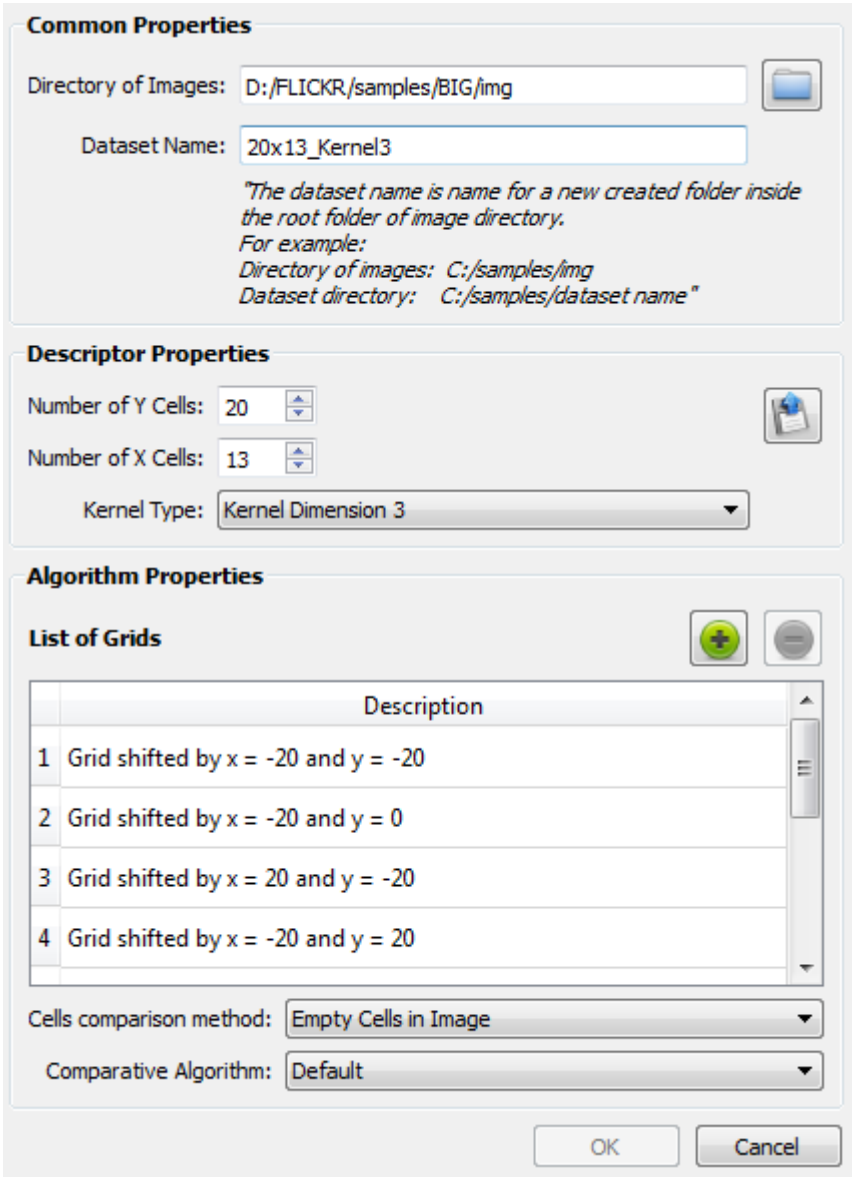
Po kliknutí na tlačítko pre načítanie datasetu sa zobrazí správa, oznamujúca, že je potrebné najprv uvoľniť aktuálny dataset v pamäti. Ako už bolo vysvetlené v podkapitole 4.2, dataset je náročný na operačnú pamäť, preto viac ako jeden načítaný dataset nemá zmysel a ani nepripadá do úvahy.

Samotná operácia načítania zaberie nejaký čas, a preto sa užívateľovi zobrazuje priebeh v progress-bare. Počas načítania sa v *Output paneli* zobrazujú súbory, ktoré už boli úspešne spracované.

5.4 Dialógové okno pre experimentálny režim

Dialógové okno má užívateľovi umožniť skúmať jak rôzne nastavenia parametrov datasetu a vyhľadávajúceho algoritmu, tak pracovať a kontrolovať výsledky dotazu získané v klasickom *Dotazovacom režime.*

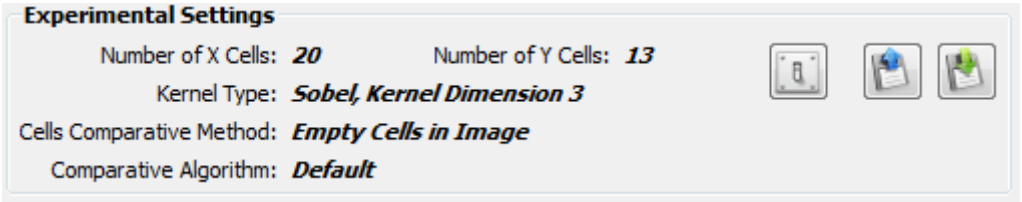
Okno sa skladá z niekoľkých komponent, ktoré možno rozdeliť do dvoch skupín podľa toho, či slúžia na zadávanie vstupu alebo na prácu s výsledkami. Hneď v hornej časti sa nachádzajú dve plátna. Ľavé plátno slúži na zadávanie dotazu a pravé na prezeranie obrázkov. Tlačítkom *Search* sa spúšťa výpočet, ktorý vytvorí deskriptory obrázkov na základe zadaných experimentálnych parametrov. Následne potom porovná deskriptory medzi náčrtkom a obrázkom.



Obr. 5.6: Dialógové okno pre vytváranie nového datasetu.

Experimental Settings komponenta je znázornená na Obr. 5.7. Zobrazuje aktuálne testovacie parametre datasetu, medzi ktoré patria rozmery mriežky a typ konvolučnej matice. Okrem toho zobrazuje informácie o nastavení algoritmu pre vyhľadávanie.

Pomocou prvého tlačítka si užívateľ zobrazí dialógové okno podobné tomu pre vytváranie datasetu. V ňom môže zmeniť nastavenia spomenutých parametrov a upraviť posúvacie mriežky. Ďalšie dve tlačítka slúžia na načítanie parametrov zo súboru alebo uloženie do súboru. Užívateľ totiž môže tieto nastavenia použiť pri vytváraní datasetu pre klasický *Dotazovací režim*.

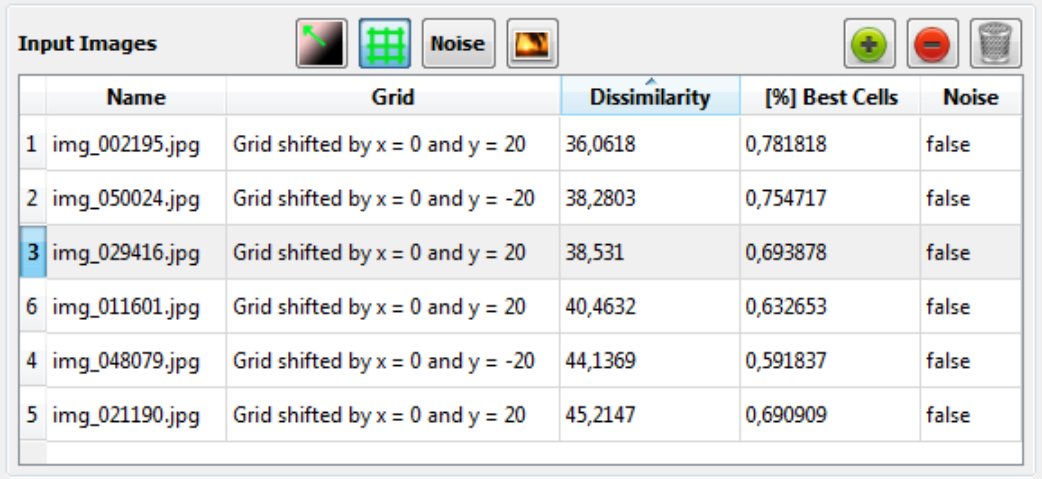


Obr. 5.7: Parametre testovaného datasetu a údaje o použitom porovnávajúcom algoritme.

Input Images komponenta je znázornená na Obr. 5.8. Má niekoľko použití naraz. Prvé z nich je predovšetkým zadávanie nových obrázkov, na ktorých sa budú testovať rôzne nastavenia datasetu. Pridávanie nových obrázkov sa robí cez tlačítko *plus*. Tlačítkom *mínus* sa zmaže vybraný riadok z tabuľky a pomocou tlačítka *kôš* sa zmažú všetky naraz.

Komponenta slúži taktiež na prezeranie výsledkov vyhľadávania. V stĺpčeku s názvom *Grid* sa zobrazuje mriežka, ktorá najlepšie pasovala pri porovnávaní daného obrázku s náčrtkom. V stĺpčeku *Dissimilarity* je najmenšie číslo podobnosti, ktoré bolo dosiahnuté pri porovnávaní s použitím jednotlivých posuvných mriežok. Stĺpček *[%] Best Cells* zobrazuje percentuálne vyjadrenie výskytu veľmi podobných buniek v danom obrázku pre najlepšiu z definovaných mriežok. Posledný stĺpček nadobúda hodnotu *true*, ak obrázok má príznak zašumeného, inak nadobúda hodnotu *false*. Kliknutím na riadok tabuľky sa zobrazí detail obrázku v pravom plátne a do tabuľky *List of Used Grids* sa načítajú dáta z výpočtu.

Ostáva spomenúť štyri tlačítka nad tabuľkou. Všetky sú vytvorené tak, aby po prvom kliknutí zostali aktívne, čo indikuje, že na vybraný riadok tabuľky sa aplikuje vlastnosť, ktorú tlačítko definuje. Ak je tlačítko aktívne, jeho vlastnosť sa aplikuje na každý nový vybraný riadok. Druhým kliknutím sa vlastnosť odoberá. Prvé tlačítko zobrazuje gradienty obrázku, druhé zobrazuje mriežku v pravom plátne. Tlačítko *Noise* je prístupné iba, ak je viditeľná mriežka v pravom plátne a jeho úlohou je modrou farbou zvýrazniť bunky, ktoré sú zašumené. Posledné tlačítko nastavuje príznak, či sa má kliknutím na riadok tabuľky zobraziť daný obrázok ako pozadie v ľavom plátne (pod náčrtkom).



	Name	Grid	Dissimilarity	[%] Best Cells	Noise
1	img_002195.jpg	Grid shifted by x = 0 and y = 20	36,0618	0,781818	false
2	img_050024.jpg	Grid shifted by x = 0 and y = -20	38,2803	0,754717	false
3	img_029416.jpg	Grid shifted by x = 0 and y = 20	38,531	0,693878	false
6	img_011601.jpg	Grid shifted by x = 0 and y = 20	40,4632	0,632653	false
4	img_048079.jpg	Grid shifted by x = 0 and y = -20	44,1369	0,591837	false
5	img_021190.jpg	Grid shifted by x = 0 and y = 20	45,2147	0,690909	false

Obr. 5.8: Tabuľka s obrázkami, na ktorých sa testujú parametre datasetu.

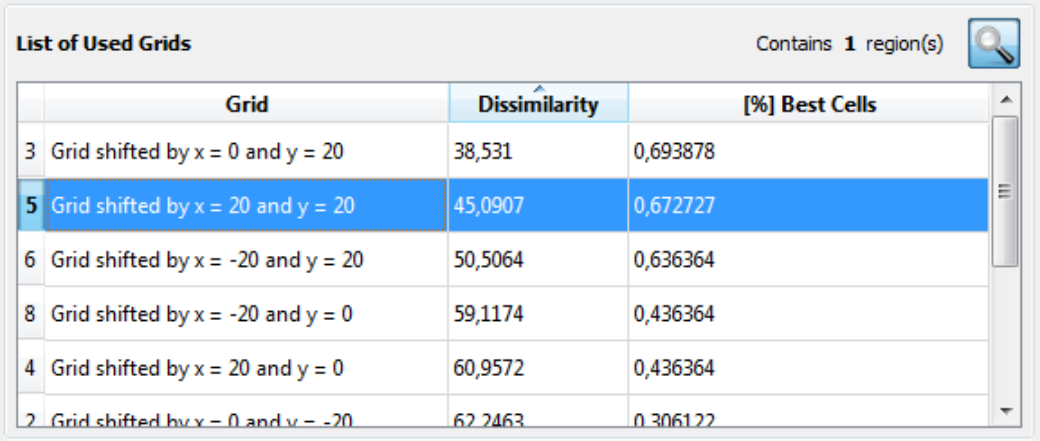
List of Used Grids komponenta je znázornená na Obr. 5.9. Obsahuje tabuľku, ktorá sa naplní po vybraní riadku z tabuľky *Input Images*. Tabuľka obsahuje toľko riadkov, koľko posúvacích mriežok bolo definovaných. Teda pre každú posúvaciu mriežku zobrazuje informácie o tom, ako dobre pasuje na obrázok, ktorý bol vybraný z tabuľky *Input Images*. Túto informáciu zobrazuje v stĺpčeku *Dissimilarity*. V stĺpčeku *[%] Best Cells* je percentuálne vyjadrenie výskytu veľmi podobných buniek.

Nad tabuľkou sa nachádza zaškrtávacie tlačítko, ktoré je prístupné len v prípade, ak je vybraný riadok tabuľky. Po kliknutí naň sa na plátne vľavo modrou farbou zvýraznia bunky, ktoré obsahujú gradienty. Zároveň sa spočíta počet rôznych regiónov, z ktorých sa skladá náčrtok. Región je množina buniek, ktoré spolu susedia buď zhora, zdola, zľava alebo zprava.

Klikaním po riadkoch tabuľky sa zároveň posúva daná mriežka po plátne pre náčrtok.

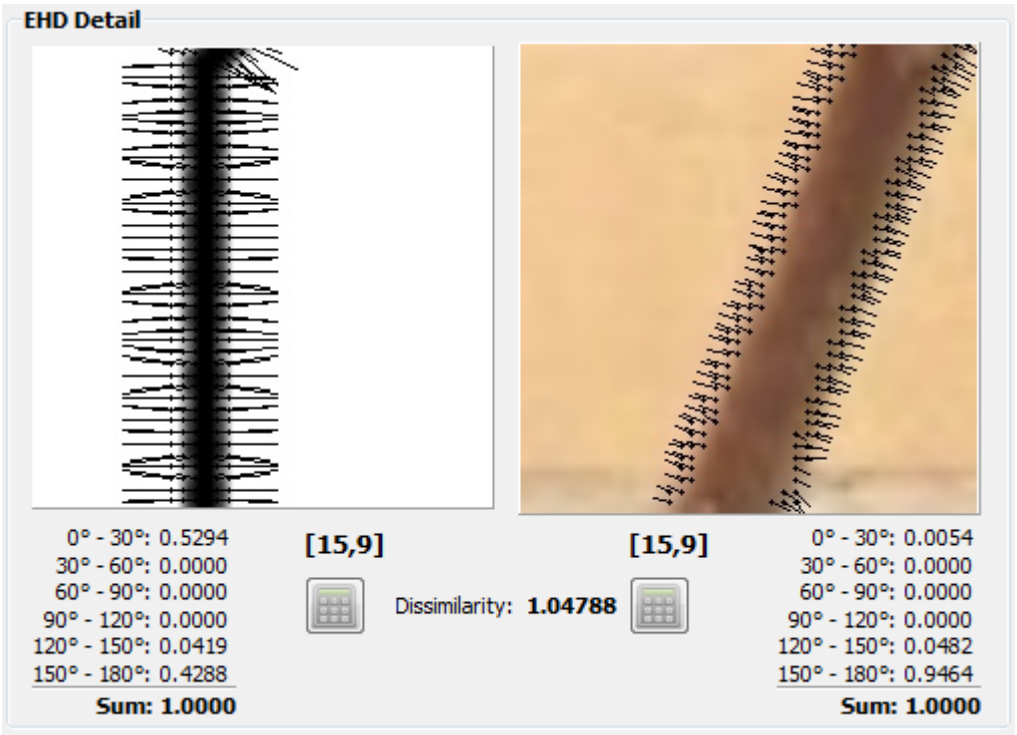
EHD Details komponenta je zobrazená na Obr. 5.10. Ide o významnú komponentu, ktorá zobrazuje detailne obsah vybranej bunky z mriežky. Na ľavej strane sa zobrazujú detaily z buniek náčrtku a na pravej strane detaily buniek z obrázku. Nech je zobrazená mriežka na obrázku, tak kliknutím stredného tlačítka myšky (prípadne kolieskom myšky) na niektorú validnú bunku[2] sa zobrazí detail tejto bunky v pravej časti. K dispozícii je zväčšený obrázok bunky s jej vyobrazenými gradientami. Ak zájdeme myškou na obrázok, zobrazí sa počet gradientov v danej bunke (ide o gradienty, ktorých veľkosť je viac ako 5% maximálnej veľkosti). Ďalšie údaje o bunke sa zobrazujú pod obrázkom – súradnice bunky a histogram znázorňujúci hodnoty v košoch. Pod histogramom je kontrolný súčet všetkých košov, ktorý musí byť vždy 1.

[2]Rozmery mriežky môžu byť stanovené tak, že pravý okraj obrázku alebo aj spodná časť obrázku môžu ostať nepokryté. V týchto miestach sú neplatné bunky. Jednoduchým zájdením myšky nad bunku sa zobrazí buď jej súradnica alebo prázdne súradnice, ktoré indikujú nevalidnú bunku.



Obr. 5.9: Tabuľka so zoznamom mriežok s rôznymi posunutiami. Pre každú mriežku je spočítané, nakoľko je podobná testovanému obrázku.

Vezmime si tlačítko znázorňujúce kalkulačku pre náčrtkovú časť komponenty. Po kliknutí naň sa automaticky získajú a zobrazia detailné informácie o bunke na tej istej súradnici, ako je súradnica obrázkovej bunky. Zároveň sa spočíta vzdialenosť týchto dvoch buniek podľa aktuálneho algoritmu pre porovnávanie buniek. Podobne funguje aj rovnaké tlačítko pre obrázkovú bunku.



Obr. 5.10: Panel s detailnými informáciami o bunke na konkrétnej pozícii.

Kapitola 6

Experimenty

Táto kapitola je venovaná experimentom vykonaným na reálnej implementácii SBIR systému. V závere budú zhrnuté výsledky a bude vysvetlené, na aký typ databáz je systém vhodný a ako treba zadávať dotazy, aby bol užívateľ spokojný s výsledkom.

6.1 Testovacie prostredie

Aby sme mohli porovnať naše výsledky experimentov s experimentmi z článku [6], potrebovali by sme k dispozícii rovnakú databázu. Keďže to nebolo možné, vytvorili sme aspoň databázu z podobného zdroja. Zo servera *Flickr* bolo automaticky stiahnutých 111 222 obrázkov. Tie tvoria databázu, na ktorej boli experimenty realizované. Fotografie vyobrazujú prevažne budovy, prírodu, ľudí a detaily rôznych objektov.

Obrázky bolo treba najprv stiahnuť a potom automaticky spracovať na rozmer 800×533. Obmedzili sme sa na tento rozmer, pretože väčšina stiahnutých obrázkov mala podobný pomer strán. Užívateľské rozhranie aplikácie bolo implementované tak, že dokáže s týmto rozmerom spoľahlivo pracovať. Iné rozmery neboli testované a nie je zaručené, že zobrazovanie bude fungovať správne.

Na disku zaberajú fotografie 13GB a priemerná veľkosť JPEG súboru je 120Kb. Testovacia zostava obsahovala procesor Intel Pentium Dual-Core 2,8GHz a operačnú pamäť 4GB. Vytvorenie datasetu z 111 222 obrázkov zaberie približne 60 až 71 minút. Čas závisí od zvolenej konvolučnej matice a rozmerov mriežky. Čím väčšia matica a čím jemnejšia mriežka, tým dlhší výpočet. Načítanie do pamäte takého datasetu sa tiež líši, ale spravidla je to v rozpätí od 40 do 70 sekúnd. Rýchlosť dotazu závisí jak od komplexnosti užívateľského náčrtku tak od rozmerov mriežky a časy sa pohybujú od 4 po 20 sekúnd. Keď je užívateľský náčrtok riedky (tj. pokrýva málo buniek), väčšina z buniek je vyfiltrovaná a počet vzdialeností, ktoré treba spočítať na jednom obrázku, je menší, čo sa vo výsledku prejaví ako rýchlejšie vyriešený dotaz.

Pre zaujímavosť, veľkosť datasetu s parametrami mriežky 20×13 a konvolučnou maticou veľkosti 5 nad databázou obrázkov veľkosti 111 222, je približne 658MB. Implementovaný algoritmus dokáže na testovacom prostredí

spracovať približne 25 obrázkov za sekundu počas vytvárania tohto datasetu.

6.2 Metódy testovania

Cieľom experimentu je overiť najlepšie parametre datasetu, t.j. rozmery mriežky a typ konvolučnej matice, zistiť, aký je vzťah medzi rôznymi typmi náčrtkov a nastaveniami datasetu.

Testované sú veľkosti mriežky 16×10 a 20×13. Testované konvolučné matice sú typu 1, 3, 5, 7. Pre každú veľkosť mriežky a typ matice bol vygenerovaný dataset. Vzniklo tak 8 testovacích datasetov. Zadefinovaných bolo 8 posunutých mriežok, odvodených od základnej mriežky s posunutím do ôsmych smerov. Posúvalo sa vždy po 20 pixeloch. Spolu tak bolo 9 mriežok aj so základnou.

Ďalej bolo vytvorených 10 rôznych náčrtkov. Pri ich výbere sa dbalo na to, aby dataset obsahoval obrázky podobné náčrtku. Zároveň boli náčrtky volené, tak aby bolo možné overiť, na aký typ dotazov je SBIR systém vhodný. Náčrtky by sa preto dali rozdeliť do dvoch kategórií: náčrtky s hranatými líniami a náčrtky s oblými líniami.

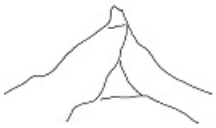
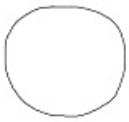
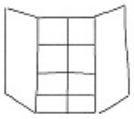
Počas testovania bol každý z vytvorených náčrtkov položený ako dotaz do každého datasetu. Následne bolo z výsledkov dotazu subjektívne vybraných niekoľko podobných obrázkov k náčrtku. Počet obrázkov bol zaznamenaný do príslušného políčka v tabuľke výsledkov (viď. Tabuľky 6.1 a 6.2). Špeciálne pre dva posledné náčrtky znázorňujúce strom a stromy bolo testované, ako ovplyvňuje výsledky vyhľadávanie v zašumených obrázkoch. Ako už bolo spomenuté, zašumené bunky relatívne spoľahlivo detekujú obrázky, kde sa nachádzajú stromy, kríky alebo listy. Predpoklad, že pri použití zašumených obrázkov bude vo výsledku viacej podobných obrázkov k náčrtkom stromu, bol v závere overený.

6.3 Výsledky testovania

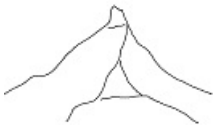
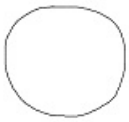
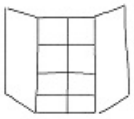
Je nutné si uvedomiť, že výsledky testovania sú podložené iba subjektívnym dojmom. Na druhú stranu, práve subjektívne pocity užívateľa vypovedajú o kvalite SBIR systému.

Po chvíľke skúmania výsledkov je možné si všimnúť, že domčeky sú lepšie rozpoznávané väčšou konvolučnou maticou a navyše, použitím jemnejšej mriežky sa kvalita rozpoznávania zvyšuje. Podobné pozorovanie sa dá vysloviť aj pre náčrtok kruhu či okna. U ostatných náčrtkov tento trend nie je jednoznačný, ale zanedbaním odchýliek by sa dalo vydedukovať podobné pozorovanie.

Dôvod lepšieho vyhľadávania je viacmenej zrejmý a už bol niekoľkokrát spomenutý. Použitím väčšej konvolučnej matice sa vygeneruje viacej gradientov, ktoré presnejšie definujú histogramy buniek. Použitím jemnejších mriežok sa zasa lepšie popíše celý obsah obrázku. Tento trend ale neplatí do nekonečna. Dá sa predpokladať, že použitím ešte jemnejších mriežok sa od istého okamihu zhoršia výsledky. Problém je práve v tom, ako užívateľ zadáva dotaz. Pri malých rozmeroch buniek sa zvyšuje pravdepodobnosť, že pri kreslení sa

Dotazy	**16×10 Kernel 1**	**16×10 Kernel 3**	**16×10 Kernel 5**	**16×10 Kernel 7**
	16	19	22	27
	29	31	30	28
	6	13	12	11
	51	52	53	56
	7	4	13	14
	12	12	12	12
	3	4	4	5
	6	3	6	5
	šumové: 14 nešumové: 6	šumové: 12 nešumové: 2	šumové: 16 nešumové: 7	šumové: 17 nešumové: 8
	šumové: 21 nešumové: 4	šumové: 16 nešumové: 9	šumové: 19 nešumové: 8	šumové: 15 nešumové: 9

Tabuľka 6.1: Tabuľka s výsledkami experimentu pre datasety s mriežkou veľkosti 16×10 a konvolučnými maticami typu 1, 3, 5, 7.

Dotazy	**20×13 Kernel 1**	**20×13 Kernel 3**	**20×13 Kernel 5**	**20×13 Kernel 7**
	21	14	24	27
	31	24	23	25
	4	11	10	10
	52	59	54	54
	16	12	14	15
	11	10	8	10
	4	4	5	5
	5	6	6	7
	šumové: 16 nešumové: 6	šumové: 13 nešumové: 4	šumové: 15 nešumové: 6	šumové: 12 nešumové: 9
	šumové: 22 nešumové: 10	šumové: 19 nešumové: 9	šumové: 19 nešumové: 11	šumové: 19 nešumové: 13

Tabuľka 6.2: Tabuľka s výsledkami experimentu pre datasety s mriežkou veľkosti 20×13 a konvolučnými maticami typu 1, 3, 5, 7.

čiara netrafí do správnej bunky a ani posúvanie mriežky túto chybu nevykompenzuje. To znamená, že najjemnejšie mriežky vyžadujú, aby užívateľ kreslil náčrtok presnejšie.

Od experimentu sa očakávalo nájdenie vhodného nastavenia datasetu pre dotazy podobného charakteru ako vytvorené testovacie náčrtky. Z výsledkov ale vyplýva, že pre zvolené náčrtky neexistuje univerzálne nastavenie. Je vidieť, že čo je dobré pre náčrtok kruhu, nemusí byť vhodné pre náčrtok domčeka. Kruh bol najlepšie vyhľadaný pri rozmeroch mriežky 16×10 a konvolučnej matici typu 3, ale domček bol naopak vyhľadaný v tomto datasete najhoršie.

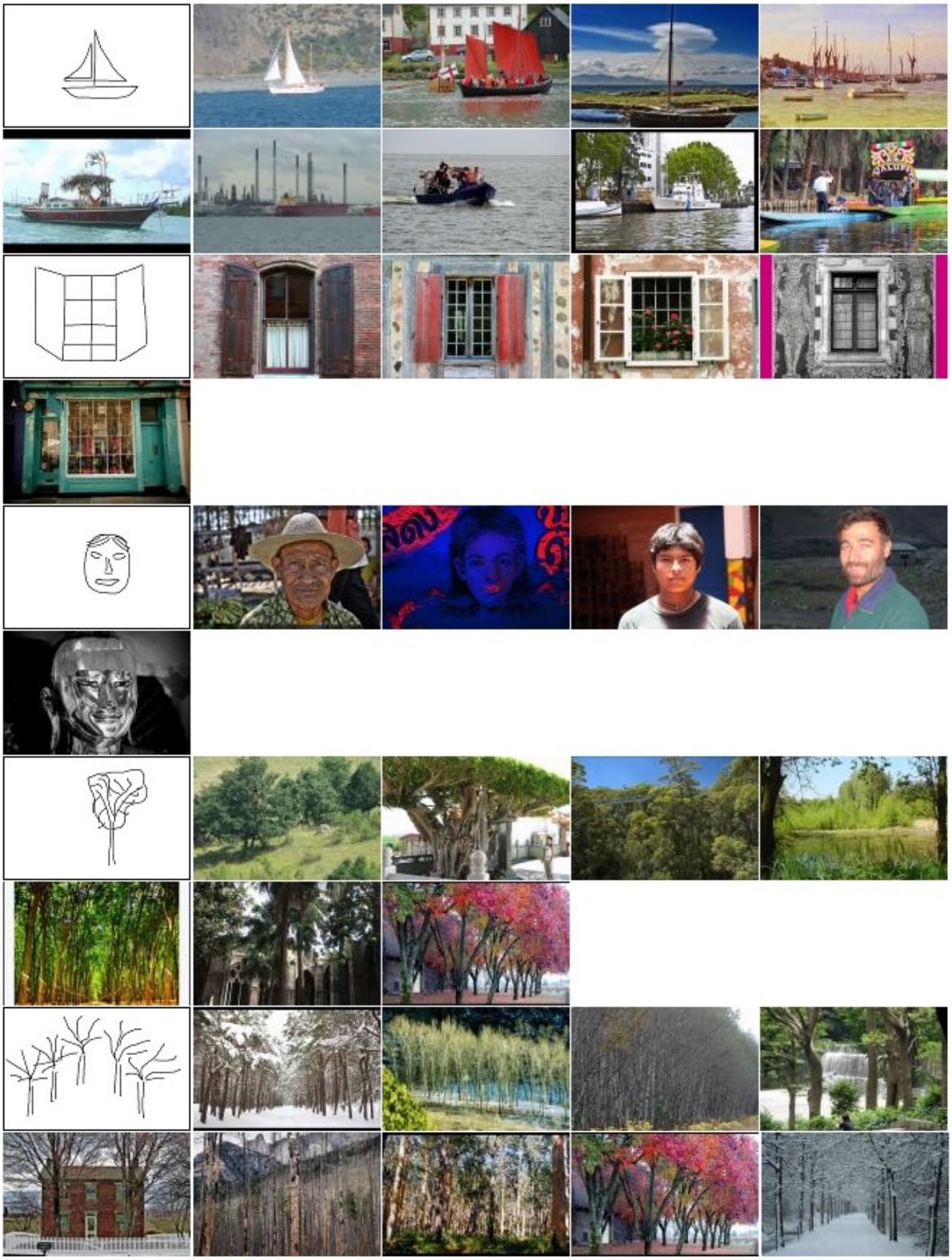
Dá sa povedať, že z výsledkov experimentu najlepšie výsledky poskytoval dataset s mriežkou 16×10 a konvolučnou maticou veľkosti 7. Keďže práve tento dataset vracal najviac podobných výsledkov, viď. Tabuľka 6.1 a pre porovnanie Tabuľka 6.2. Predpokladáme však, že pre iného užívateľa by boli vhodnejšie iné parametre datasetu. Prehľad výsledkov, ktoré vrátil najlepší dataset, je vidieť na Obr. 6.1 a Obr. 6.2. Na obrázkoch je vidieť niekoľko subjektívne vybraných najviac sa podobajúcich obrázkov k danému náčrtku. Výber bol robený z výsledku dotazu, ktorý vracal obrázky v počte 60 až 80. Na priloženom DVD je možné v adresáre *Experiments* vidieť všetky výsledky z testov.

Počas vývoja aplikácie a jej testovania sme prišli k záveru, že na zadávanie vstupného náčrtku by bolo vhodnejšie použiť tablet s kresliacou plochou. Kreslenie myškou je pomerne nepresné a často sa stáva, že užívateľovi skĺzne ruka. Oblé tvary ako napríklad kruh, kvetina, tvár sa myškou veľmi ťažko kreslia. Ale aj napriek tomuto obmedzeniu poskytuje aplikácia zaujímavé výsledky.

Implementovaná architektúra bola testovaná iba na databáze rôznych fotografií z dovoleniek, ktoré obsahovali prírodu, mestá, budovy, more, niektoré známe architektonické pamiatky, postavy a len zopár objektov, ktoré boli fotené spredu alebo zboku. Väčšina užívateľov sa však snaží nakresliť objekt práve z tohto uhlu pohľadu (napríklad auto zboku, alebo dom spredu), a preto sa do výsledku dostane iba málo skutočne podobných objektov (z jednoduchého dôvodu, pretože databáza neobsahuje objekty s podobnou siluetou ako náčrtok).

Pred zadávaním dotazu si musí užívateľ uvedomiť, aký typ náčrtku môže nakresliť. Nie je možné očakávať, že po zadaní náčrtku s detailným nákresom auta z perspektívy sa vo výsledku zobrazí podobný obrázok. Dôvod je jednoduchý a vyplýva z myšlienky, na ktorej je založená celá architektúra. Náčrtok musí obsahovať predovšetkým hlavné, výrazné línie. Zbytočné detaily v náčrtku často zhoršujú výsledok. Predstavme si situáciu, keď na obrázku je auto a mriežka má také parametre, že pokryje toto auto štyrmi bunkami. Vzhľadom k tomu, aký komplexný objekt je auto, je možné očakávať, že histogramy týchto štyroch buniek bude len veľmi obtiažne napodobniť v náčrtku. Majme teraz opačnú situáciu, a to obrázok domu, ktorý zaberá podstatne viac buniek ako spomenuté auto. Dom má jasnú hranatú siluetu, ktorá sa dá veľmi jednoducho reprodukovať v náčrtku. Dokonca aj napriek chybám, ktoré užívateľ zakreslí do náčrtku, je jeho deskriptor veľmi podobný väčšine domov v datasete.

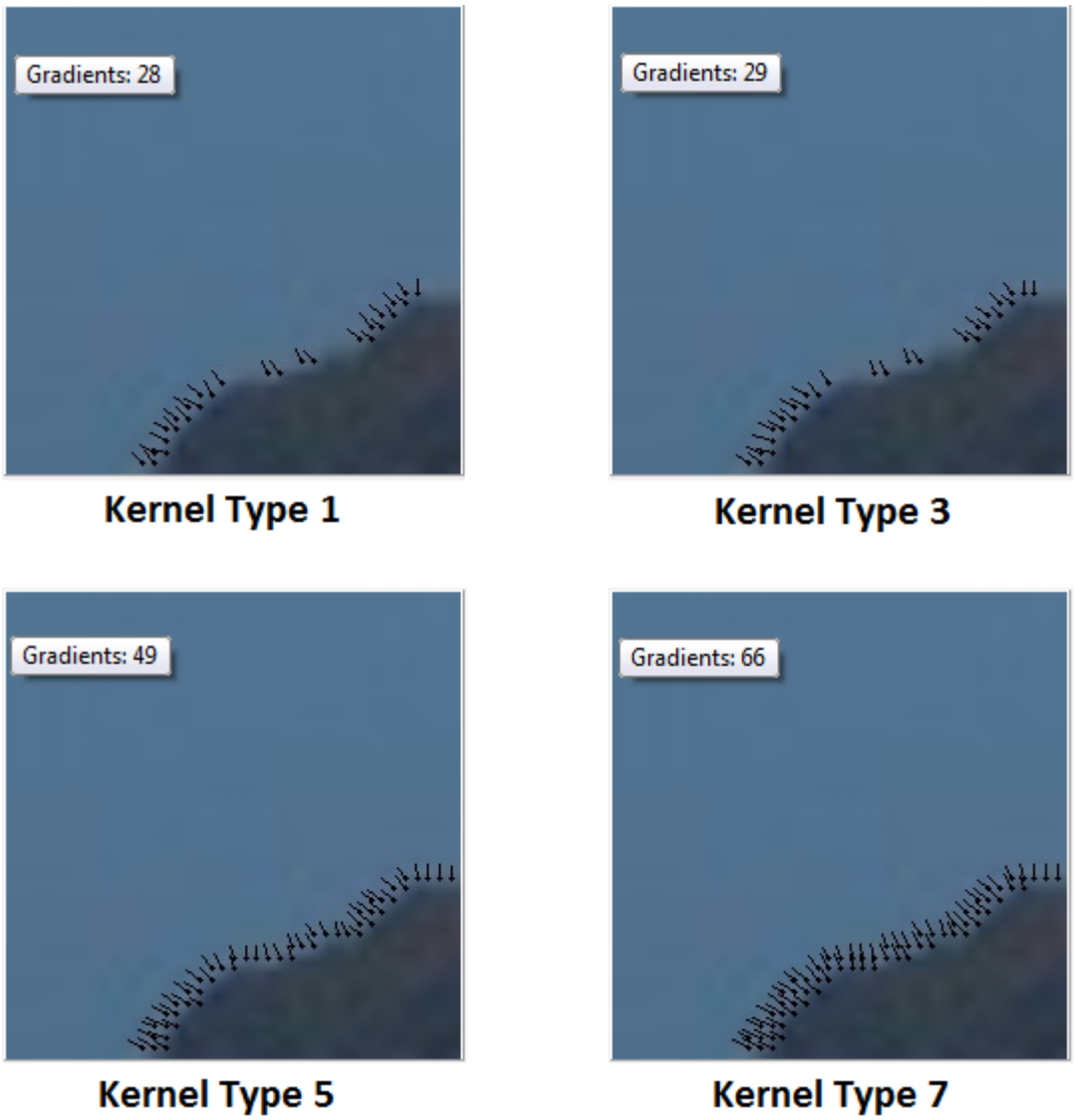
Obr. 6.1: Niektoré najlepšie výsledky získané z datasetu s mriežkou 16×10 a konvolučnou maticou veľkosti 7.



Obr. 6.2: Niektoré najlepšie výsledky získané z datasetu s mriežkou 16×10 a konvolučnou maticou veľkosti 7.

6.3.1 Konvolučné matice

Použitie väčších konvolučných matíc spôsobuje spomalenie offline výpočtu, čo je pochopiteľné, keďže väčšia matica vyžaduje viacej operácií na spočítanie gradientu v danom bode. Overili sme, že väčšia matica generuje gradienty aj tam, kde menšia nie, ako príklad viď. Obr. 6.3. Táto vlastnosť vyhovuje dovtedy, kým tieto nové gradienty nespôsobujú, že sa bunka stane zašumenou. V takom prípade je vhodné použiť menšie matice.

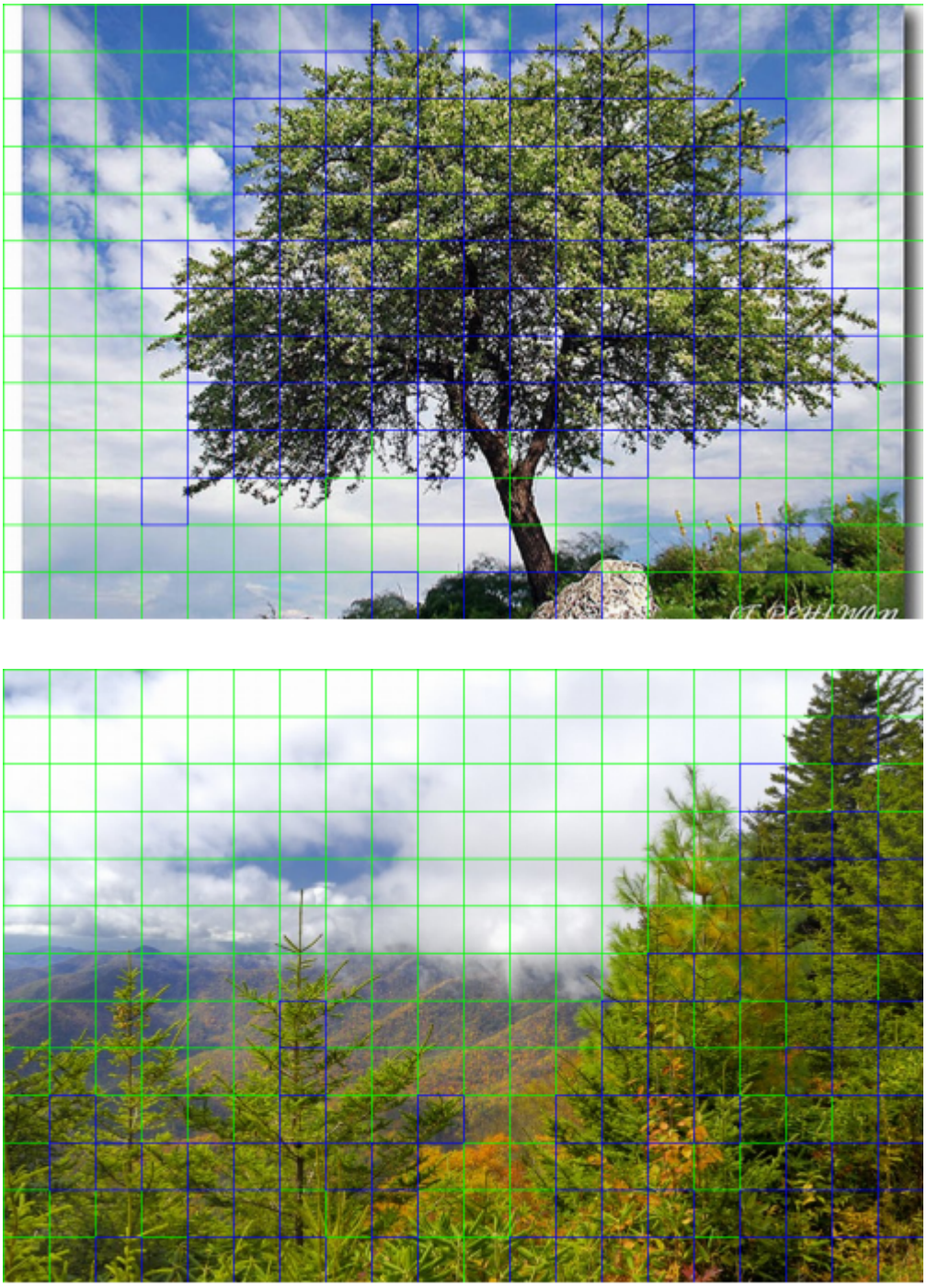


Obr. 6.3: Znázornené vypočítané gradienty tej istej bunky pomocou rôznych konvolučných matíc. V hornom ľavom rohu každého obrázku je zobrazený počet gradientov v bunke.

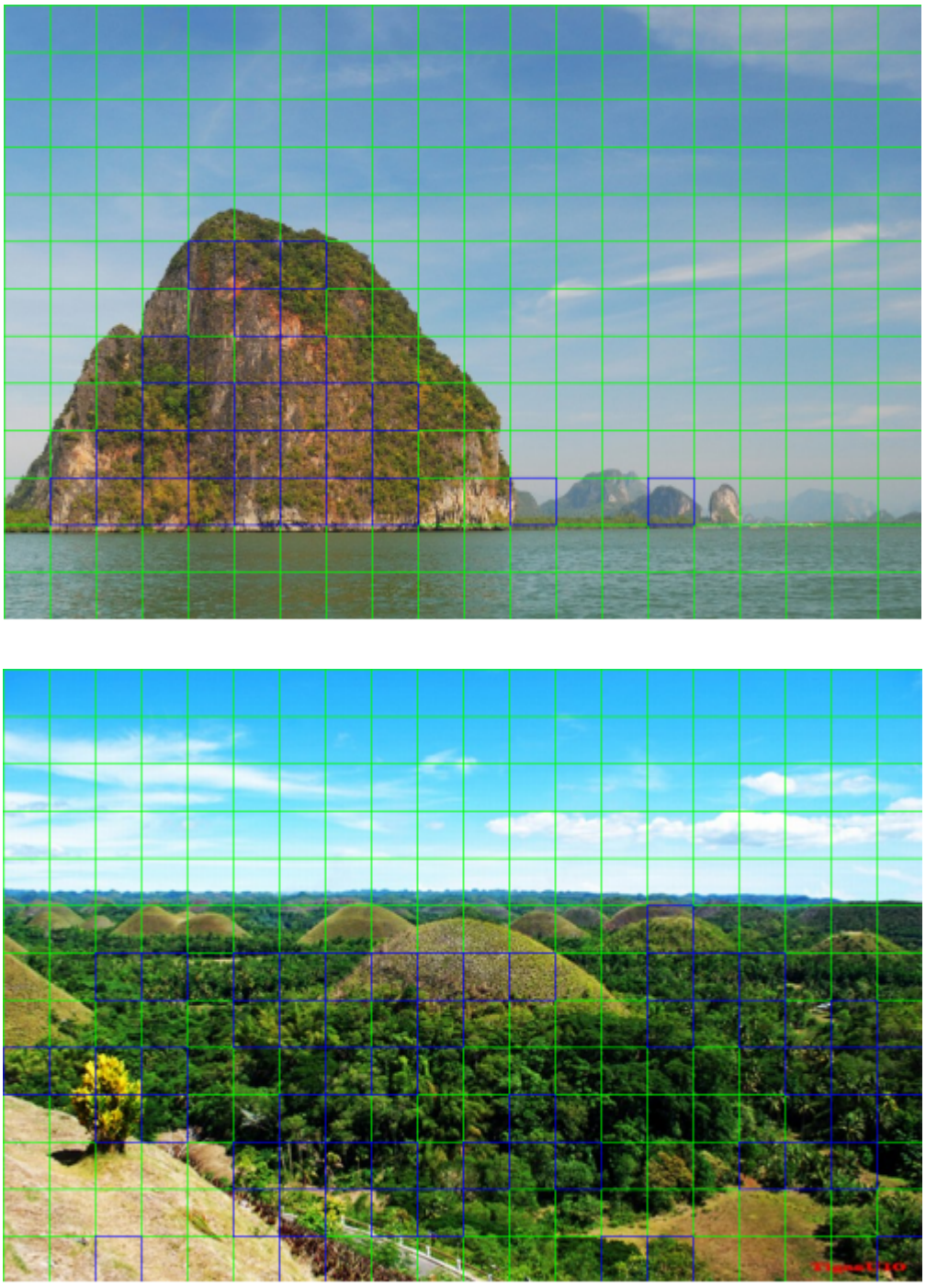
6.3.2 Rozpoznávanie neštruktúrovaných objektov

Tak, ako sa predpokladalo, stromy možno lepšie nájsť v zašumených obrázkoch. Vlastnosť *zašumených buniek* otvára nové možnosti použitia implementovaného deskriptoru. Identifikovaním takýchto buniek, je možné detekovať na obrázku časť stromu, kríky, listy alebo iné nesymetrické, neštruktúrované objekty z prírody. Z výsledkov experimentov vyplýva, že stromy sú jednoduchšie detekovateľné v obrázkoch, o ktorých sa vie, že sú zašumené (tj. viac ako 15% ich buniek s gradientami sú považované za zašumené).

Vďaka deskriptoru a *zašumeným bunkám* by sme mohli byť schopní rozpoznávať obrázky so stromami. Okrem toho by bolo možné ich dostatočne presne lokalizovať. Vybrali sme pár obrázkov (viď. Obr. 6.4 a 6.4), na ktorých je znázornené, že ide o celkom spoľahlivú techniku rozpoznávania. Pre kvalitnejšie použitie tejto techniky hľadania a lokalizácie stromov je nutné hlbšie skúmanie, ktoré by však bolo nad rámec tejto práce.



Obr. 6.4: Ukážka rozpoznávania stromu. Modré štvorčeky identifikujú časti stromu, prípadne listy.



Obr. 6.5: Ukážka rozpoznávania objektov z prírody, na obrázku hore bol identifikovaný kopec, pretože obsahuje kríky a malé stromy. Na dolnom obrázku boli rozpoznané zelené plochy, ktoré opäť zodpovedajú stromom alebo kríkom. Modré štvorčeky identifikujú časti stromu, prípadne listy a trávu.

Kapitola 7

Záver

Cieľom práce bolo navrhnúť a implementovať architektúru pre získavanie obrázkov z databázy na základe rukou nakresleného dotazu. Samotné vyhľadávanie a získavanie podobných obrázkov bolo založené na porovnávaní deskriptoru z obrázku a deskriptoru z užívateľského náčrtku. Ide o vylepšenú verziu deskriptoru, navrhnutého pre MPEG-7, ktorý je známy ako *Edge histogram descriptor*. Jeho základom je popis lokálnej distribúcie hrán.

Bol zavedený pojem *dataset* ako abstrakcia nad databázou obrázkov. Vytvára sa počas offline procesu, v ktorom má za úlohu automaticky detekovať výrazné hrany (línie) na obrázkoch a popísať ich pomocou hranového deskriptoru. Každý obrázok je rozdelený na bunky, v ktorých sa počíta histogram hrán. Histogram sa spočíta z gradientov, ktoré definujú hranu, keďže gradient je vlastne normála k priamke v danom bode.

Deskriptor a algoritmus bol reiplementovaný podľa článku [6]. Po analýze jeho vlastností boli navrhnuté vylepšenia, ktorých funkčnosť sa následne overila v reálnej implementácii. Keďže výsledok vyhľadávania závisí od kvality nakresleného dotazu, cieľom vylepšení bolo zmenšiť túto závislosť. Jedným z návrhov na zlepšenie je *posunutá mriežka*, ktorá má kompenzovať chybu v náčrtku v prípade, že sa užívateľ netrafí do správnej bunky. Identifikácia *prázdnych buniek* a *veľmi podobných buniek*, slúžiacich na preusporiadanie výsledku, doplňuje zoznam vylepšení.

Za významný prínos sa pokladá schopnosť algoritmu detekovať *zašumené bunky*, ktoré poslúžia k identifikovaniu obrázkov obsahujúcich časť stromu, kríky, listy alebo iné nesymetrické, neštruktúrované objekty z prírody.

V experimentálnej časti bol skúmaný vplyv parametrov datasetu na výsledky. Pre vybranú množinu náčrtkov vrátil najlepšie výsledky dataset s mriežkou 16×10 a konvolučnou maticou veľkosti 7. Získané výsledky z experimentu demonštrujú, že implementovaný systém poskytuje užívateľovi intuitívny prístup k databáze obrázkov.

Ďalej z experimentov vyplýva, že dobre sa hľadajú útvary, ktoré sú hranaté ako dom a stĺpy. Je to zapríčinené tým, že užívateľ vie lepšie nakresliť objekt hranatý ako napodobniť oblé tvary (loď, kopec). Avšak kruh je výnimkou, ako je vidieť z výsledkov, pretože dataset obsahuje veľa takýchto obrázkov.

Jemné mriežky lepšie popisujú obsah obrázku, ale pre užívateľa je náročnejšie

nakresliť dotaz tak, aby dostal podobné výsledky. Na druhej strane mriežky s veľkými bunkami kompenzujú chyby v náčrtku, ale popisujú obrázok chudobnejšie.

Implementovaný deskriptor je rýchly pre vyhodnotenie podobností, avšak jeho invariantnosť voči merítku, rotácii a posunutiu je obmedzená. Táto nevýhoda je kompenzovaná kompaktnosťou deskriptoru a možnosťou ho použiť na veľké databázy obrázkov.

Dotazy zadávané užívateľom by nemali obsahovať malé detaily v rámci jednej bunky, ale namiesto toho výrazné línie cez niekoľko buniek. Taktiež pred zadávaním dotazu je potrebné si uvedomiť či v miestach, kde bude čiara nakreslená, môžu existovať gradienty. Nakreslením čiary na miesto kde gradienty nie sú v hľadanom obrázku spôsobí posunutie tohoto obrázku na nízke pozície prípadne sa vôbec nezobrazí vo výsledku.

7.1 Ďalší vývoj

Ako prvý krok pre ďalší vývoj by bolo vhodné získať rádovo väčšiu databázu obrázkov o počte niekoľko miliónov, na ktorej by sa ukázali nedostatky v optimalizácii rýchlosti a kompaktnosti dátových štruktúr. Ďalej by bolo vhodné zrušiť obmedzenie na veľkosť obrázkov, ktoré môže databáza obsahovať a ponechať iba obmedzenie na pomer strán.

Ďalej by bolo zaujímavé otestovať systém nad databázou kreslených obrázkov. Predpokladá sa že nad takouto databázou by implementovaná architektúra dosahovala ešte lepšie výsledky, keďže kreslené obrázky sa viac podobajú náčrtku ktorý je tiež kreslený.

Z implementačného hľadiska je triedenie vo vyhľadávaní neoptimálne a v prípade používania skutočne rozsiahlych databáz obrázkov by samotné vyhľadávanie trvalo dlho. Jednou z navrhovaných zlepšení je držať si po celú dobu hľadania k najlepších výsledkov, čím by odpadla úloha zotriediť veľký zoznam čísel.

Ako už bolo spomenuté, *zašumené bunky* v kombinácii s informáciou o farbe v daných bunkách by mohli byť použité pre identifikovanie obrázkov obsahujúce stromy, kríky alebo obecne prírodu. Okrem toho by sa dali tieto objekty na obrázku presne lokalizovať. Samozrejme ide o hypotézu, ktorá vyžaduje ďalšie skúmanie.

Odkazy na DVD

[i] Verzia aplikácie použitá pri experimentoch popísaných v kapitole 6. Obsahuje navyše iba tlačítka pre uloženie náčrtku, načítanie náčrtku a uloženie výsledkov dotazu.
`CD:\Application\ForExperiments`

[i] Verzia aplikácie spustiteľná bez inštalácie. Upozornenie: V niektorých prípadoch bez inštalácie nebude aplikácia bežať korektne.
`CD:\Application\Release`

[i] Inštalačný súbor aplikácie.
`CD:\Application\Setup`

[ii] Adresár obsahuje výsledky z experimentov. Z názvu podadresárov sa dá odvodiť zodpovedajúci použitý dataset a náčrtok.
`CD:\Experiments`

[iii] Databáza 10 000 obrázkov, na ktorej je možné testovať aplikáciu.
`CD:\Images\Database10k`

[iii] Databáza 500 obrázkov, na ktorej je možné vykonať rýchle testy aplikácie.
`CD:\Images\Database500`

[iii] Databáza vybraných obrázkov, na ktorých sa dá skúmať správanie sa zašumených buniek.
`CD:\Images\ImagesWithNoise`

[iv] Zdojové kódy aplikácie.
`CD:\Sources`

[iv] PDF súbor diplomovej práce
`CD:\Text`

[iv] Ukážkové video práce s aplikáciou.
`CD:\Video`

Literatúra

[1] Blanchette J.; Summerfield M.: *C++ GUI Programming with Qt 4, Second Edition*, Prentice Hall, February 04, 2008

[2] Bradski G., Kaehler A.: *Learning OpenCV, Computer Vision with the OpenCV Library*, O'REILLY, First Edition, September 2008.

[3] Byoung Chul Ko, Hyeran Byun: *Query-by-Gesture: An Alternative Content-Based Image Retrieval Query Scheme*, Journal of Visual Languages and Computing (2002) 13, strany 375-390

[4] Chee Sun Won, Dong Kwon Park, and Soo-Jun Park: *Efficient Use of MPEG-7 Edge Histogram Descriptor*, ETRI Journal, Volume 24, Number 1, February 2002.

[5] Eakins J., Graham M.: *Content-based Image Retrieval*, University of Northumbria at Newcastle, Report: 39, October 1999

[6] Eitz M., Hildebrand K., Boubekeur T. and Alexa M.: *A descriptor for large scale image retrieval based on sketched feature lines*, EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling, 2009

[7] Green B.: *Edge Detection Tutorial, 2002*, `http://www.pages.drexel.edu/~weg22/edge.html`

[8] Green B.: *Canny Edge Detection Tutorial, 2002*, `http://www.pages.drexel.edu/~weg22/can_tut.html`

[9] Sobel I., Feldman G.: *A 3x3 Isotropic Gradient Operator for Image Processing*, Machine Vision for Three-Dimensional Scenes, strany 376–379. Academic Press, 1990.

[10] Wikipédia: *Feature detection (copmuter vision)*, `http://en.wikipedia.org/wiki/Feature_detection_(computer_vision)`